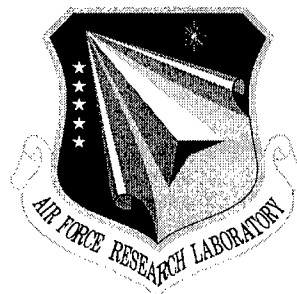


AFRL-IF-RS-TR-2001-242

Final Technical Report

November 2001



REAL TIME DYNAMIC LANGUAGES

iRobot Corporation

**Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D900**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

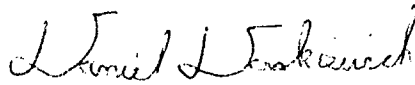
20020308 042

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

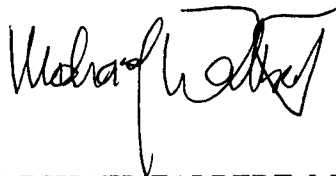
AFRL-IF-RS-TR-2001-242 has been reviewed and is approved for publication.

APPROVED:



DANIEL E. DASKIEWICH
Project Engineer

FOR THE DIRECTOR:



MICHAEL TALBERT, Maj., USAF, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/ITB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REAL TIME DYNAMIC LANGUAGES

Polly Pook, John Aspinall,
Gill Pratt, Rodney Brooks,
and Arturio Arsenio

Contractor: iRobot Corporation
Contract Number: F30602-96-C-0280
Effective Date of Contract: 26 August 1996
Contract Expiration Date: 29 December 2000
Short Title of Work: Real Time Dynamic Languages
Period of Work Covered: Aug 96 - Dec 00

Principal Investigator: Polly Pook
Phone: (617) 629-0055
AFRL Project Engineer: Daniel E. Daskiewicz
Phone: (315) 330-7731

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION
UNLIMITED.

This research was supported by the Defense Advanced Research
Projects Agency of the Department of Defense and was monitored
by Daniel E. Daskiewicz, AFRL/ITB, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE NOVEMBER 2001		3. REPORT TYPE AND DATES COVERED Final Aug 96 - Dec 00
4. TITLE AND SUBTITLE REAL TIME DYNAMIC LANGUAGES			5. FUNDING NUMBERS C - F30602-96-C-0280 PE - 62702F PR - D900 TA - 01 WU - 01	
6. AUTHOR(S) Polly Pook, John Aspinall, Gill Pratt, Rodney Brooks, and Arturio Arsenio				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) iRobot Corporation 22 McGrath Highway Twin City Office Center, Suite #6 Somerville Massachusetts 02143			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-242	
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Daniel E. Daskiewich/IFTB/(315) 330-7731				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) A behavioral programming approach to adaptive autonomous control had been completed. An advanced hexapedal crab-like robot, Ariel, was built in order to investigate methods of robotic self-adaption. As a result of this program, Ariel is able to negotiate the harsh surfzone using fully autonomous distributed gait control. Advancements include the implementation of a high-level controller that recognizes impending unstable etrajectories in the phase space of the gait, and adapts accordingly. Separately, a primate-like head was developed for multi-layer visual control of bipedal and quadrapedal robot navigation. The software architecture incorporates cognitive motivations, selective attention, and a suite of reflexive visual routines. the real-time adaptive software architecture developed under this effort is generically applicable to and DoD embedded real-time system.				
14. SUBJECT TERMS Real Time Software Architecture, Adaptive Software, Autonomous Robots			15. NUMBER OF PAGES 52	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

Section I.	Executive Summary	1
I.1	CLIN 0003 - SAFER	1
I.2	CLIN 0004 – Active Vision Head.....	2
Section II.	CLIN 0003: A Behavioral Programming Approach to Adaptive Autonomous Control (SAFER)	3
II.1	Ariel 2 Hardware Overview.....	4
II.1.1	Mechanical Architecture	4
II.1.2	Electrical Architecture.....	6
II.2	Software Infrastructure.....	10
II.2.1	Behavioral Programming and Subsumption Architectures	10
II.2.2	L/MARS	11
II.2.3	RTEMS	11
II.2.4	Mirror Interprocessor Communications Substrate	11
II.3	Adaptive Gait Control (or Making a Crab Think like a Stick-Insect).....	13
II.3.1	Background	13
II.3.2	Overview of the Stick Insect Controller	14
II.3.3	Using the Stick Insect Controller on Ariel.....	15
II.3.4	The Stick-Insect Controller as a Dynamic System.....	15
II.4	Results.....	25
II.4.1	Straight Line Motion	25
II.4.2	Turning Motion.....	26
II.5	Conclusions	26
II.6	References	27
Section III.	CLIN 0004: Adding an Active Vision Head to the M4 Robot.....	28
III.1	Introduction	28
III.2	Body/Head Integration	29
III.3	Hardware Architecture	30
III.3.1	Design and Manufacturing.....	31
III.4	Software Architecture.....	37

Table of Figures

Figure 1: Ariel 1, a crab-like robot for the surfzone.....	3
Figure 2: The mechanical architecture of Ariel.....	4
Figure 3: One of Ariel's six identical legs.	5
Figure 4: Cutaway view of Ariel foot showing pressure sensor	6
Figure 5: The computational architecture of Ariel.	7
Figure 6: A comparison of FarNet and COTS networks.	9
Figure 7: The software architecture of Ariel.	12
Figure 8: Leg geometry for a hexapod.....	13
Figure 9: The individual leg cycle.	14
Figure 10: The 2-Leg Phase Space	16
Figure 11: 2 Leg System with 1 Influence	17
Figure 12: 2-Leg System with Influence 5.....	17
Figure 13: Limiting Behavior for Influence 5.....	18
Figure 14: Other results for 2 Leg Systems.....	18
Figure 15: Alternative Phase Space Representation: Unfolded, Uniform Time Spacing for a Two-Oscillator System.....	19
Figure 16: 3-leg System shown as a 3-torus unfolded as a cube	20
Figure 17: Good trajectory.....	20
Figure 18: The linear controller (one per joint — 12 total)	22
Figure 19: The trajectory.	23
Figure 20: Upper Level Control.....	24
Figure 21: Real world data: Leg Position for Straight Motion	26
Figure 22: Macaco - M4/M2 Project. Rodney Brooks and Artur Arsenio. Primary Sponsor: Air Force Aerospace Research – OSR. Secondary sponsor: iRobot (IS Robotics) contract number F30602-96-C-0280.	28
Figure 23: M4/M2 Head Hardware Organization	31
Figure 24: M4 (dog-like head) design.....	32
Figure 25: M4 solid model.	32
Figure 26: M2 (primate like head) design, with four parts replaced, and a stereolithography artwork. Top: Chimpanzee face anatomy. Middle: Primate head skeleton (four parts replaced). Down: Final primate design.....	33
Figure 27: M4/M2 Components.....	34
Figure 28: Active binocular tracking (1 st version).....	39
Figure 29: Smooth pursuit tested on Cog.....	39
Figure 30: Approaching postures (curiosity behaviors - left), and object tracking (right).	40

Section I. Executive Summary

The Real-Time Dynamic Languages contract consists of three separable efforts:

1. Evolutionary Design of Complex Systems (EDCS) (CLINs 0001 and 0002).
2. A Behavioral Programming Approach to Adaptive Autonomous Control (SAFER) (CLIN 0003).
3. Adding an Active Vision Head to the M4 Robot (CLIN 0004).

The first effort, EDCS, was completed and a final report submitted previously. This document includes the final reports for each of the latter two efforts, SAFER and Active Vision for the M4 Robot.

I.1 CLIN 0003 - SAFER

The goal of the SAFER program is to explore roles and methods for self-adaptive software. Robotics is an ideal field for such exploration, as robots are independent entities that must adapt to changes in their environment (or internal failures) if they are to perform for any practical duration. For the SAFER program, we have adapted the amphibious *Ariel* robot. Ariel is modeled on the crab and is a proof-of-concept for mine remediation on the beach and in the surf-zone. On six legs, Ariel must be able to traverse sand, underwater silt, rocks and pebbles, while withstanding crashing waves, undertows and water currents. Thus, Ariel has particular need for self-adaptive gait control.

An initial prototype of Ariel was built under a grant from the Office of Naval Research. The SAFER program has funded:

- the development of a second Ariel prototype;
- the evaluation of a COTS networking system for robots, Echelon LONWorks™.
- the application of an innovative, proprietary, real-time networking system, FarNet™.
- the application and extension of the Cruse "stick insect model" of distributed gaited motion on Ariel
- the development of a novel high-level controller, built on top of the Cruse model, for self-adaptation.

The results are promising for autonomous robots in extremely taxing conditions. The new Ariel performed a walking pattern search in the North Atlantic surfzone (Revere Beach in Revere, Massachusetts), at the Coastal Systems Service in Panama City, Florida (at a demonstration sponsored by Boeing Aerospace), and at Monterey Bay for National Geographic. Ariel II is able to withstand pressures at a minimum of 25 feet of depth, and carry a payload of 6 kg.

The COTS Echelon LONWorks™ networking system was found to be not up to the task of reactive robot control, contrary to marketing claims. The Echelon system lacks predictable latencies, rendering poor performance for a distributed control models. It also has generally slow messaging that prevents good centralized control. Thus neither a distributed controller, such as the Cruse stick insect model, nor a Centralized Pattern Generator could be built on top of LONWorks™ for hexahedral walking.

We replaced Echelon with the just-developed proprietary iRobot FarNet™ system with great success. The system has low, predictable latencies and high bandwidth. Thanks to the SAFER program, we were able to perform final debugging on the system and turn FarNet™ into a commercial success. FarNet™ is now included in nearly all iRobot robot sales.

The Cruse distributed model of gait control was implemented on Ariel and critically extended to skidding and turning. It became apparent that the model could only perform reactive functions with slow

convergence on to a stable gait after perturbation or failure. What is required is a higher-level controller that can perform two key functions. The first is to recognize unstable trajectories and effectively knock the controller back into a limit cycle. This is roughly akin to our ability to catch our step instantly after stumbling, rather than gradually change the step in small successive increments until stable again. The second role is to recognize the need for a change in gait, such as when the terrain changes.

We developed two models for higher level adaptive control. The first looks to return map analysis for recognition of trajectories as they become unstable. The second is to perform clustering on sensory signals to recognize terrain change. The first has promising results; the second, at this point, is a thought model and so not presented here. Both warrant future work.

1.2 CLIN 0004 – Active Vision Head

The MIT AI Lab, in a subcontract to iRobot, developed a head (eyes and neck) for bipedal and quadrupedal robots along with a suite of visual routines. The physical design is inspired by primates and canines and is called, alternately, the Macao (for the primate), the M2 (a two-legged robot) or the M4 (a four-legged robot) head.

Macaco is a 7DOF small and light robotic head, with four color CMOS cameras and one thermal camera. This head will be incorporated into a moving body. The merging of social competencies with navigation capabilities requires an architecture that integrates all the modules coherently, such that Macaco is given a personality demarcated by curiosity and a wish of interacting with people, together with a strong instinct for safety.

Macaco was designed for navigation in unstructured environments. Thus, its vision system must comprise methods of assessing the constantly changing terrain. Navigation for obstacles and potentially treacherous landscape avoidance, slope detection, and gaze stabilization are all requisites for such competence. Furthermore, in order to be a convincing social participant, the vision system must also allow for person detection and inference of human gaze direction. All of these vision modules must also be accessible to each other and concurrent with other sensor input from the rest of the body. The software performs three parallel functions. The function of the top layer is cognitive decision-making. The second layer attends to events selected by the cognitive layer. The third layer performs reflexive actions, such as gaze-tracking and smooth pursuit

The work continues beyond the scope of this contract. The final report is thus cast in ongoing terms.

Section II. CLIN 0003: A Behavioral Programming Approach to Adaptive Autonomous Control (SAFER)

A behavioral approach to robot control software can add a degree of robustness to environmental complexity that is difficult to achieve using more static controller-design methods. Self-adaptive software techniques can add a similar layer of robustness along with compact representation. We have taken a behavioral approach to self-adaptive software to provide compact, robust, and decentralized control for a walking robot designed for operation in quite dynamic and unpredictable environments.

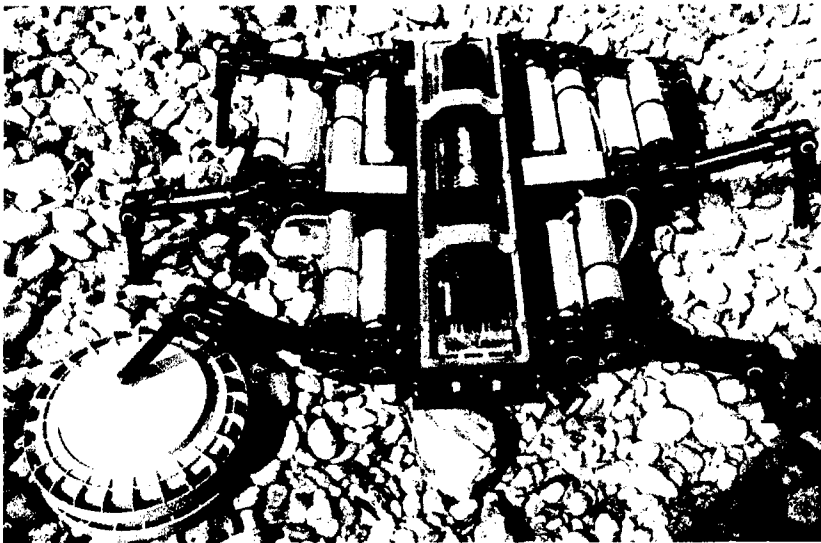


Figure 1: Ariel 1, a crab-like robot for the surfzone.

Ariel is a six-legged autonomous underwater robot designed for mine remediation in the surf zone. This existing platform had several different walking controllers but suffered from a lack of robustness and responsiveness in the complex dynamics of the surf zone environment. The key goal has been the preservation of the real-time responsiveness so necessary for control in the real world while enabling smarter, more adaptive behaviors to provide a robust yet simple control over the broad range of anticipated environmental contexts. The need for responsiveness makes it unacceptable to stop the robot's locomotion in order to perform small modifications to the walking controller. This means that any adaptation mechanism must work on an active set of lower-level behaviors.

Our starting point for the development of a decentralized walking controller was an existing controller for stick-insect-like gaits. This has been adapted to the novel situation of crab-like, sideways motion to provide additional stability in the complex dynamics of the surf zone. An additional walking controller was also developed to maintain posture during turn-in-place maneuvers, where the high degree of slippage imposes additional dynamic constraints.

This behavioral approach to the walking controllers also provides the opportunity for self-adaptation at higher levels. We have taken two approaches: extracting simplified predictive models of the walking controllers for use at the higher level and generation of additional contextual knowledge of the environment through better exploitation of sensory assets. Additional analysis is aimed at quantifying the cost of decentralized control for the Ariel system.

The ideal (unperturbed by the environment) walking controller can be seen as a simple dynamic system. The upper layer of software can make short-range predictions about the trajectory of the lower-layer controller in the configuration space and, in particular, can predict entering undesirable regions of configuration space with enough time to prevent instabilities. Our intent is for the upper layer to intervene in a minimal way to alter the behavior of the lower layer. In robot-specific terms, undesirable regions of the configuration space are when adjacent legs both lift in return stroke at the same time. The consequence is mechanical instability and stumbling in the gait. Minimal intervention consists of slowing the gait or commanding a leg to enter return stroke early. As mentioned above, brute force intervention such as freezing the motion by interrupting the controller is contrary to our goals. Results indicate increased robustness in the walking controller for both straight-line and turning maneuvers.

The second approach to enabling higher-layer monitoring is the extraction of additional knowledge from sensory data that is used by the lower layer. This information may consist of correlations, clusters, or other long-time-scale measurements. Our ultimate intent is to use this information to modifying the lower layer in response to environmental changes. We have achieved some success in building the foundations, in both software and hardware, for generating information that can be more effectively exploited by the higher layers through adaptation of the lower-layer walking controllers. In robot-specific terms, the lower layer controllers use force data from the six foot-sensors only to a limited extent. The walking controller is interested only in a contact/no contact decision. However, much more information is available in both time and force-resolution. If we could, for instance, detect whether Ariel were walking on sand or a hard bottom, we could modify gait parameters accordingly. With the present advances in foot pressure sensors, compass placement, and software infrastructure, the information now in use by the low-level walking controllers can be more effectively exploited in future versions of the system. Preliminary, qualitative results indicate that surface compliance can be recognized, paving the way for further enhancements in adaptive high-level control for Ariel.

II.1 Ariel 2 Hardware Overview

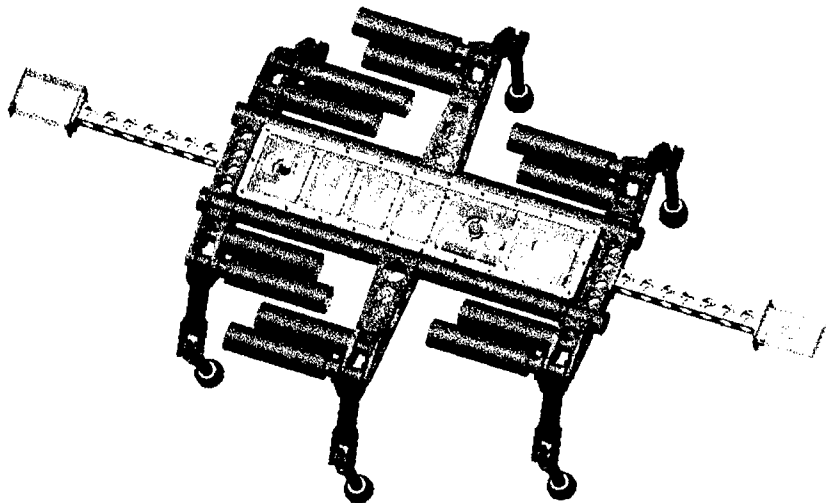


Figure 2: The mechanical architecture of Ariel.

II.1.1 Mechanical Architecture

Six leg mounts surround a central watertight housing. The main structural members run longitudinally alongside the housing double as battery containers. The outrigger arms at each end support two separate housings for compass/inclinometers. Each two degree-of-freedom leg measures 15 cm from hip to knee,

and 15 cm from knee to foot. Cable drives run from two motors mounted adjacent to each other near the hip joint, to their respective joints. Each of Ariel's 12 motors is a 20-watt, brushed DC motor. The motor gearhead is a 236:1 multi-stage planetary gearbox. The robot weighs approximately 11 kg on land, and 5 kg fully submerged.

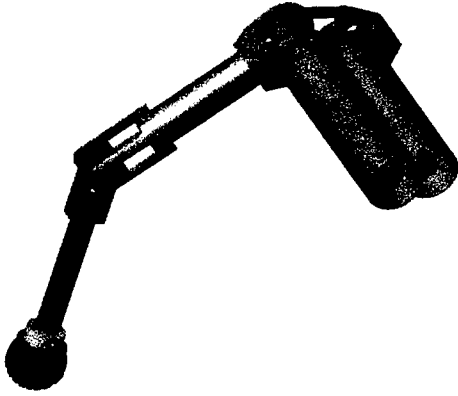


Figure 3: One of Ariel's six identical legs.

The two degree-of-freedom leg (shown in Figure 3) is activated by motors contained in individually watertight housings. Each motor is a 20-watt, DC, brushed, gearhead motor. Cable drives connect each motor's output shaft to its corresponding joint. The hip-to-knee distance is 15 cm; the knee-to-foot distance is approximately the same.

The leg design is one of the pieces of Ariel that has remained virtually unchanged since the original Ariel 1 design. The foot, however, is a new addition and is described below.

Force-Sensitive Foot

Previous versions of Ariel have employed a binary (contact/no-contact) foot switch to sense contact with the ground. While simple to interpret, the switch suffered from both mechanical and software problems. On the mechanical side, the switch was prone to fouling by sand or other debris from the surface on which Ariel was walking. On the software side, the force necessary to activate the switch was, by necessity, a fixed quantity. Since Ariel effectively weighs more than twice as much on land as it does underwater, the single force value for a threshold between contact and no contact was too restrictive.

We wanted a new foot-contact sensor satisfying the following design criteria:

- Robust against fouling by mud, sand, and other debris.
- Provide as much information about contact as possible, within constraints of the first criterion.

The resulting design is shown in Figure 4. A flexible sealed envelope assures that debris will not foul the operation of any mechanical contact. The foot is filled with distilled water. A scalar pressure sensor inside the envelope provides a measure of the force being exerted on the foot. The water fill and the differential pressure sensing combine to compensate for volume changes due to variations in the ambient external pressure. The use of scalar pressure (as opposed to resolving the force into directional components) is a compromise on the second design criterion, in order to achieve a simple and robust solution to the first.

The foot is cast from a two-part silicone rubber compound. The collar is made from aluminum and the foot is sealed to the collar using the two grooves and a wire tie.

Amplified by an op-amp in lower leg, and passing to an A/D input on the motor board, the pressure sensor provides 8-9 bits of force resolution.

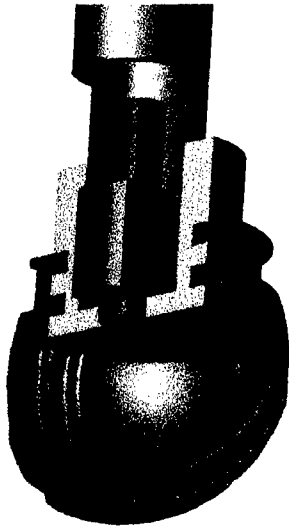


Figure 4: Cutaway view of Ariel foot showing pressure sensor

Extended Frame for Compasses

Measuring the local magnetic field, in order to get a compass heading, is a necessity for Ariel to walk a straight line as part of a search pattern. Previous experience with an internally mounted compass was unsatisfactory. Stray, fixed magnetic fields, from the permanent magnets in the motors, made a calibration procedure mandatory. Transient, magnetic fields (most likely from motor currents) were also present; these could not be “calibrated out”. To address these issues, two compasses were mounted on arms extending from the central chassis. The fall-off of the magnetic field from the motors and other electronics dictated the length of these arms. By using a symmetrical arrangement, differential measurements are also available, improving robustness through redundancy.

II.1.2 Electrical Architecture

Figure 5 is a block diagram of Ariel’s major components, and their interconnection. Ariel’s 12 motors are controlled from four motor control boards. Each board controls three motors, communicating via a proprietary ring network to the control processor. Two 16-bit processors provide computational power (described below under software). Ariel carries two compass/inclinometers at the end of outrigger arms. The arms isolate the compass measurements from large transient currents to the motors. The compass/inclinometers are commercial off-the-shelf models. A pressure sensor is surface mounted on top of the chassis and used to prevent Ariel from walking beyond its depth limit (8 m.).

Batteries

Ariel is powered from two series-connected battery packs. Each battery pack contains 11 NiCd cells for a nominal system voltage of 26.4 volts. Fully charged, they start at about 30 volts in practice. The current NiCd cells give it a charge capacity of two amp hours; the use of nickel-metal-hydride (NmH) batteries would increase the charge capacity to approximately three amp hours. Battery lifetime, in use, varies with the various environmental conditions that affect the load on the motors, but typically runs from 30 minutes to an hour.



IS ROBOTICS

Computational Architecture

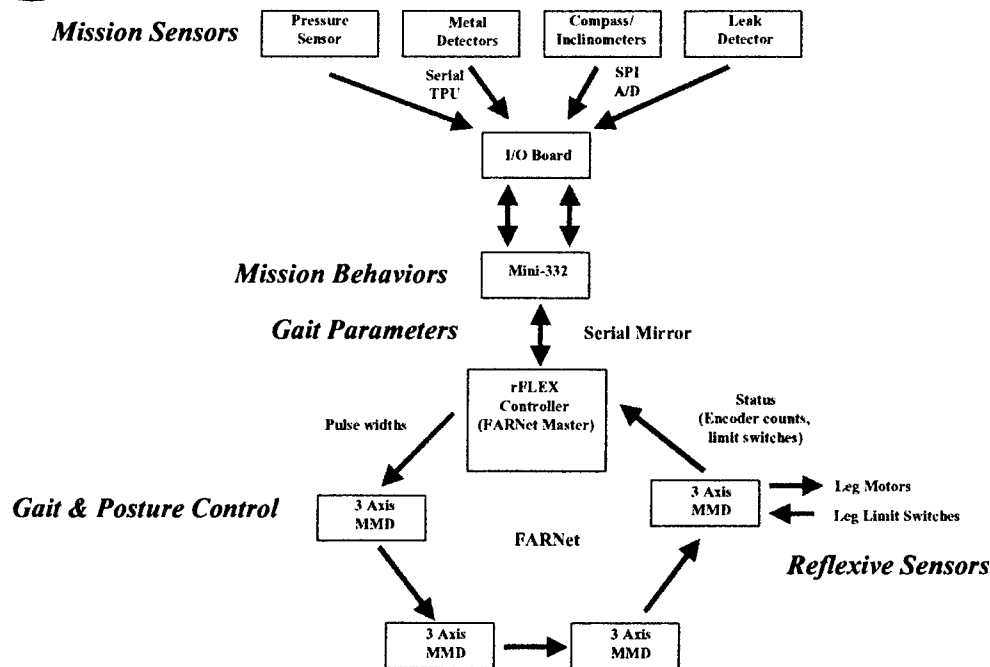


Figure 5: The computational architecture of Ariel.

LONWorks™ Network

Ariel I was controlled with a MC68332, six MCH6811, and twelve PICs. While this configuration served to evolve a fast prototype, it provided insufficient processing power for advanced software development and a cabling nightmare. Consequently, we looked to a networked computational model for Ariel II, that would provide fast, modular, distributed computation.

We initially chose to use the commercially available Echelon LONWorks™ system. The on-board computational network was distributed across a 16Mhz MC68332 processor and a network of “neuron” microcontrollers using the LONWorks™ protocol. Each Echelon neuron (a 10MHz MC143150 or MC143120) provides dedicated local processing to the sensing and actuation devices while the faster 68332 processor coordinates action and communicates with the OCU. The central and dedicated processors are arranged along a daisy-chain backbone and communicate using LONWorks™’ 7-layer protocol. With this architecture, individual sensors, activators, and boards become line-replaceable units, facilitating reconfiguration as well as debug and repair of malfunctioning robots. The hardware, moreover, reflects the distributed software framework of behaviors wired together.

iRobot wrote all of the motor and sensing drivers in Neuron C, a language for the Echelon LONWorks™ distributed processing system. The LON (Local Operating Network) is a distribution of special Motorola microcontrollers, called Neurons, with a full communications protocol stack. Each Neuron can be dedicated to a mechanical or electrical device, including robot sensors, actuators and other microprocessors, such as the Motorola 68332 or Pentium. Thus, the computational requirements of controlling the low-level robot hardware can be off-loaded from a central processor and handled individually by the dedicated Neurons. The Neurons are arranged on a bus for which LONWorks™ transparently handles all communication, creating a seamless and modular system.

LONWorks™ Results

The selection of LONWorks™ was motivated by the desire to distribute processing and the lack of a suitable distributed network for small autonomous robots. The integration of LONWorks™ with our existing MC68332 embedded robot controller was a significant effort. The results were poor. LONWorks™, despite sales claims, does not exhibit predictable latencies and their duration is typically a long 15 msec. While the advertised bandwidth is 1.2 Mbit/sec, in actuality the system throughput was 0.5Mbit/sec once communication overhead was included. This created an unworkable network for distributed gait control, wherein motor controllers depend on real-time reporting from neighboring controllers.

Concurrently, iRobot had been developing a proprietary computational network to address the needs of fast, predictable, distributed networks for robots. The result, called FarNet™, far outstripped commercially available systems. Though in a prototype stage, we chose to replace LONWorks™ with FarNet™.

FarNet™ Network

FarNet is iRobot's proprietary robot component interconnection technology. It was designed to provide fast communication with low and predictable latency among robot components. FarNet is an arbitrated ring, for predictable latency. Nodes are bit-serial for minimal delay, and are implemented using FPGA technology. The network bit rate is 16Mbit per second. By restricting the maximum packet size, we can reduce typical latencies to tens of microseconds. The FarNet ring is hardwired on Ariel because of space considerations; we expect most typical implementations to use a hub for reliability (a hub can "splice" individual nodes in and out of the ring). At the packet level, FarNet protocols admit either single-master or multiple-master operations. Single-master networks provide for less latency while multiple-master networks permit more distributed control.

FarNet™ Results

FarNet performed superbly, although the implementation of the newly created Beta version required great effort. The network incorporates a suite of motor controllers and sensors, with a special software communication "mirror" between the FarNet Coldfire processor and the MC68332 embedded robot controller. Details are sketched throughout below.



State-of-the-Art Embedded Networks

(Estimated!)

	FARnet (IS Robotics)	LON (Echelon)	CAN	IEEE-1394 (fire-wire)	USB
Type	Fast arbitrated ring	Broadcast	Broadcast	Tree	Tree
Throughput (Mbits/s) Raw Actual	16 10-12	1.2 0.5	1 0.5	400 200	12
Reliable delivery with minimal overhead	YES	NO	YES	NO	
Latency	10µsec	Msec (unpredictable)	100 µsec	Msec	Msec
Fault-tolerance?	YES	NO	NO	Partial	Partial
Size limits Cable length # of nodes	100m 100	Big Big (speed tradeoff)	Big Big (speed tradeoff)	5m 100	5m
Auto-configurable?	YES	NO	NO	NO	YES

Figure 6: A comparison of FarNet and COTS networks.

Processors

Ariel carries two processors on board. iRobot's mini rFlex processor provides low-level control (as described in the software section) to Ariel's 12 motors. The rFlex processor provides:

- Motorola Coldfire chip with 90MHz clock;
- 16Mbytes of RAM;
- 2Mbytes of "flash" EEPROM;
- 2 high speed serial ports;
- FarNet interface: bit serial link at 16 Mb/sec implemented in Xilinx Spartan series XCS30 FPGA; the FPGA is programmable at processor boot time from the nonvolatile memory.

Higher-level software runs on an iRobot mini332 processor. The miniboard provides:

- Motorola 68332 processor with a 16 MHz clock;
- 1Mbytes of RAM;
- 2Mbytes of "flash" EEPROM;
- 2 high-speed serial ports, with more available using TPU.

Additional I/O

The IO board supports eight channels for analog to digital conversion and provides four serial ports to interface between devices and the central processor. An inversion switch mounted on the board signals

the lateral orientation (belly or back side up) of Ariel so that the legs can flip about for locomotion in either position.

Compass/Inclinometers

As mentioned above under New Mechanical Hardware, Ariel carries two compass/inclinometers at the end of outrigger arms. The arms isolate the compass measurements from large transient currents to the motors. The compass/inclinometers are commercial off-the-shelf MicroStrain 3DM models. Each compass provides a measurement of the local gravity and magnetic field vectors.

Motor drivers

Ariel's 12 motors are controlled from four motor control boards. Each board controls three motors, communicating via the FarNet ring network (see above) to the control processor. The specs for each board are:

- Current capacity: 5A peak, 3A continuous at $24V \pm 6V$ (18V to 30V);
- FarNet interface: bit serial link at 16 Mb/sec implemented in Xilinx Spartan series XCS30 FPGA;
- Three independent channels of 8-bit PWM;
- Three channels of quadrature decoding;
- Three channels of current sensing;
- Power supply voltage sensing and undervoltage lockout;
- Shorted motor winding detection and protection;
- Ten bits of digital input for limit switches (or other purposes).
- Four analog inputs to 10-bit A/D converters.

II.2 Software Infrastructure

The overall software architecture is the subsumption architecture described below. As shown in Figure 5, the electrical architecture of Ariel is distributed between two processors, a 68332 and an rFlex Coldfire. The rFlex Coldfire is running the RTEMS operating system and features code developed in C for low-level control. The 68332 mini-processor is running the Venus operating system with L/MARS software for behavioral control.

II.2.1 Behavioral Programming and Subsumption Architectures

Autonomous robots provide a challenge to control system design. They are designed to venture into a world that is dynamic and unpredictable. It is impossible to test all situations that the robot may encounter. Thus, their controllers cannot depend on complete or precise knowledge of all potential stimuli.

Behavioral control is a decentralized approach to the architecture of a control system. Control is distributed among a number of asynchronous modules, or behaviors, each limited in the scope of its inputs and outputs to a small fraction of the total system state. The organization of a behavioral control program into layers of competence, where each layer is capable of operating without control from higher layers, is known as subsumption architecture.

Behavior-controlled systems have been demonstrated to be less compute-intensive and more robust than competing centralized approaches. This makes behavior control ideally suited to small mobile robotic applications where reduction in size and energy consumption dominates the design process.

Subsumption architectures are intended to be inherently adaptive. One behavior can suppress another, allowing a simple structure in which alternative algorithms can reside in different behaviors without interfering with each other.

The key to adaptability in a subsumption architecture is that high-level redundancy, *i.e.* the ability to complete some task or detect some condition by multiple means, does not necessarily imply lower-level redundancy, *i.e.* that the different means are simply copies of the same mechanism. For example, a multiple-legged robot may have different gaits that achieve the goal of walking. However, the individual legs may not be identical, the different gaits may not employ the same numbers of legs, nor may they result in a walk with the same speed, efficiency, or stability. Nonetheless, as long as they achieve the goal of locomotion, a system that adapts to use them will display a more graceful degradation than one that doesn't.

II.2.2 L/MARS

The portable Common Lisp subset used to develop the software is based on previous work by iRobot on L, a Common Lisp subset that currently runs on 68000-based machines. L was carefully designed and implemented for use in small, embedded processors operating under real-time constraints; it is currently in use on a number of walking, tracked, and wheeled robots being developed by iRobot.

The memory management system for L has been redesigned, making use of recent research results by the garbage collection community, particularly in the area of real-time garbage collection. This permits the relaxation or elimination of certain restrictions on programs imposed by the L language or its previous implementation.

The process-oriented language is based on previous work by iRobot on MARS (Multiple Agency Reactivity System). MARS is a system built on top of the multi-threading features of L, and is written entirely in L. MARS provides mechanisms for defining, creating, and manipulating structured collections of threads which share parts of a lexical environment in arbitrarily complex ways, along with mechanisms for adding and removing communication links between these threads.

The development environment is written entirely in Common Lisp, but the graphical user interface and the serial communications facility for communicating to a target system are both written using implementation specific extensions provided by Macintosh Common Lisp.

II.2.3 RTEMS

RTEMS is an open source, portable, standards-based real-time executive for C, C++, and Ada95 originally developed by On-Line Applications Research Corporation for the U.S. Army Missile Command. RTEMS complies with the POSIX and RTEID/ORKID standards. It provides support for TCP/IP, remote or local thread aware debugging using gdb, and file system support. Basic kernel features include multitasking, homogeneous and heterogeneous multiprocessor systems, event-driven, priority-based, preemptive scheduling, optional rate monotonic scheduling, intertask communication and synchronization, priority inheritance, responsive interrupt management, and dynamic memory allocation. RTEMS provides a high level of user configurability, and is currently available on a large array of microprocessors. Access to the RTEMS source code allowed us to make minor modifications tailoring the real time executive to our specific needs.

II.2.4 Mirror Interprocessor Communications Substrate

The mirror layer implements a "mirroring of variables" model on top of serial streams. Mirroring of variables (as opposed to remote procedure call, or an event model) was chosen for compatibility with existing software.

The rFlex processor runs the RTEMS real time operating system. Code is developed in the C language for the linear controller and gait controller levels. The mini332 runs the Venus operating system. Code is developed in iRobot's L language, and in MARS, a macro package written in L. The purpose of the mirror layer is to communicate between various threads of control running on the different processors shown in Figure 7.

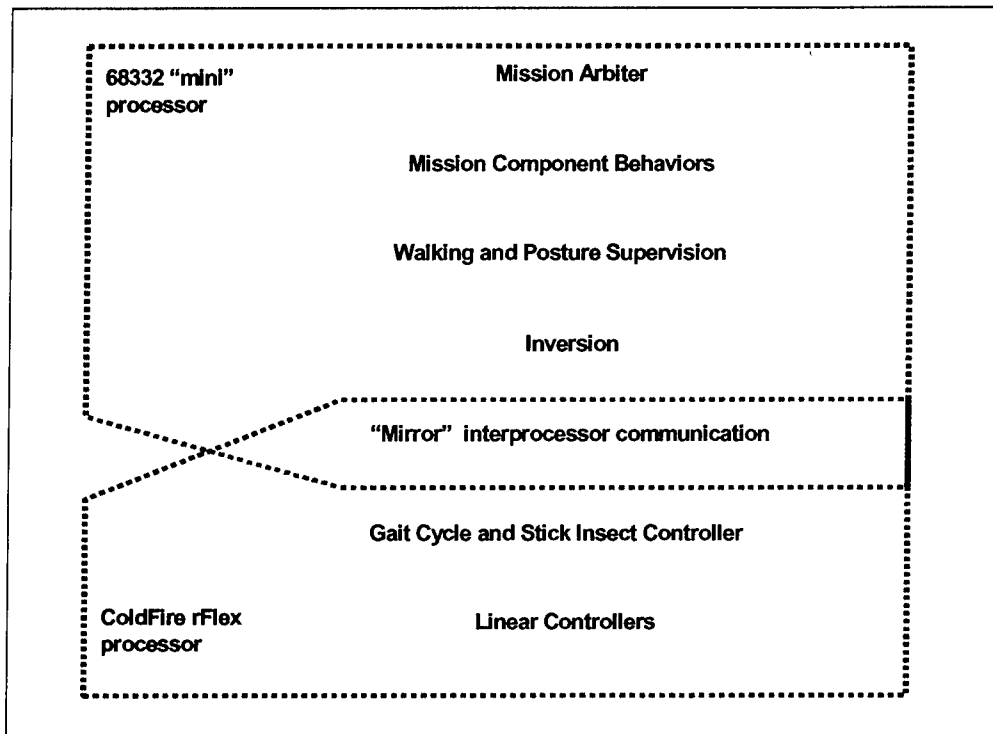


Figure 7: The software architecture of Ariel.

The multiple pieces of software that implement the mirror (.c, .h, and .lisp files) are written from one interface description shared among several robot projects, thus fulfilling a key requirement of code for the mirror layer that consistency be maintained between code on the two sides of the inter-processor connection. The interface description is written in a domain specific embedded language which is read by a code generator that generates both L code for the mini68332 side and C code for the rFlex Coldfire side of the communication link. The mirror layer design allows the software to be independent of the underlying communications medium.

Finite State Machine Language

The Finite State Machine (FSM) language is a Domain Specific Micro-Language embedded in L and developed under this project. It allows a programmer to express the description of an augmented finite state machine, with states and transitions clearly delineated from regular L code.

This FSM language has since been reused by another iRobot commercial project. The FSM is at the heart of the mission control software of the BoreRat robot. The BoreRat is a first-of-its-kind untethered device for taking tools and sensors down into oil wells and then back to the surface again under its own power and control. All other approaches to oil well data gathering and maintenance currently require spools of wire or coiled tubing to push the devices in and pull

them out, with each spool weighing several tons. By eliminating the need for these spools for some kinds of oil well interventions, we can significantly reduce the cost of these operations and eliminate the dangerous task of handling this heavy equipment. The BoreRat was first tested at Baker Hughes' water filled test well in Bossier City, Louisiana. When it returned to the surface from a depth of 356 ft on March 15, 1999, this was the first time any device had ever come back up out of an oil well under its own power. As well as being used to control the queues of tasks to be performed, health monitoring, and robust control to overcome stuck conditions, each of the low level tasks it knows how to do is also executed as an FSM until just above the PID-based velocity controllers.

II.3 Adaptive Gait Control (or Making a Crab Think like a Stick-Insect)

One of the primary goals of the SAFER program was to develop a more robust legged locomotion algorithm for Ariel. A distributed, self-adaptive approach was chosen. The stick-insect walking controller [Crus95] was modified and extended for the crab-like motion characteristic of the Ariel platform. Crab-like motion was originally chosen for the Ariel platform for biomimetic reasons: the crab is the most efficient walker in surf-zone regions.

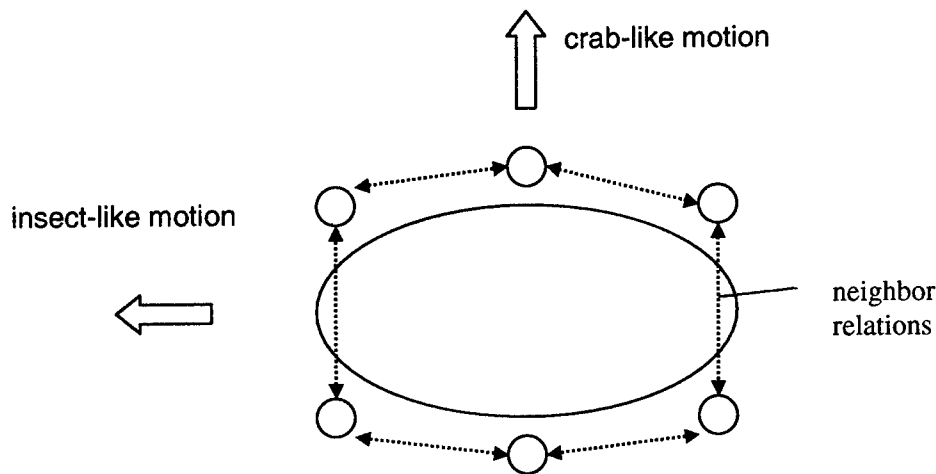


Figure 8: Leg geometry for a hexapod.

II.3.1 Background

The goal of any legged locomotion controller is to move the legs of a robot in a coordinated fashion to achieve the following goals:

Mobility: forward progress is made;

Mechanical Stability: the robot doesn't fall down.

It is usual to define *static* mechanical stability, as mechanical stability where inertial dynamics play no part in the motion. Such mechanical stability is achieved by keeping at least three legs on the ground so that the center of mass falls inside the enclosing polygon of the legs currently in contact.

For a hexapod robot, the above criterion for mechanical stability is equivalent to a simpler description: **No two adjacent legs should leave the ground at the same time.** Legs are considered adjacent if they fall next to each other in a circular ordering, as shown in Figure 8.

It is important to realize that a locomotion controller is more than a specification of the ideal trajectory of all the legs. The controller must also specify what the legs should do in a non-ideal situation. It must

deal robustly with perturbations away from the desired state, as such perturbations are an inevitable consequence of operating in the real world.

II.3.2 Overview of the Stick Insect Controller

The stick insect controller can be subdivided into:

- The trajectory of an individual leg;
- Inter-leg coordination.

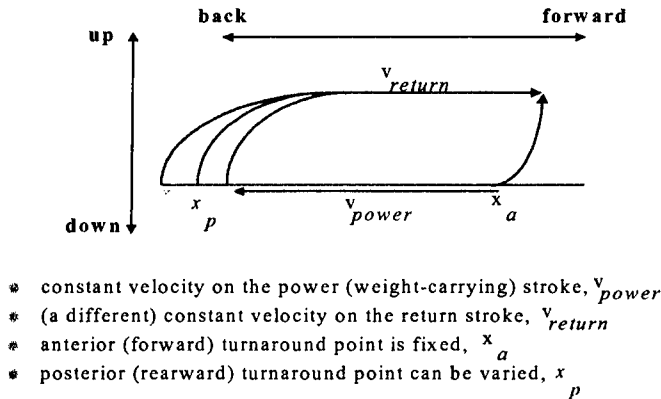


Figure 9: The individual leg cycle.

Individual leg motion is shown in Figure 9. Each leg executes a cycle consisting of a power stroke from front (anterior) to back (posterior) of the robot, and a return stroke from back to front. The power stroke is where the robot's weight is carried forward with the foot in contact with the ground. The power stroke maintains a constant velocity from start to end; this is the velocity of the robot over the terrain. The return stroke also maintains a constant velocity - usually larger than the power stroke velocity. The anterior turnaround point is a fixed parameter. The posterior turnaround point varies, in response to signals (called *influences*, in the literature) from other legs. This is the mechanism whereby one leg can shorten or lengthen the time for a complete cycle, and therefore change its phase relationship with another leg.

In normal operation, the return velocity is fixed at a large value, and the power velocity is chosen to be any value less than or equal to the return velocity. The ratio of the two velocities determines how many legs, on average, will be in contact with the ground. A larger power velocity means fewer legs, on average, in contact. Therefore, there is a tradeoff between speed and stability.

Inter-leg coordination is achieved by influences (as defined immediately above) between neighboring legs. A neighbor is adjacent on the same side, or immediately opposite, as shown in Figure 10 below.

An influence extends, or retards, the posterior turnaround point of the *receiving* leg as a function of the current position of the *sending* leg.

It is important to note what the stick insect controller does *not* contain. It does not contain any single centralized authority for timing. Individual legs participate in the controller as peers, and coordinated motion emerges as a consequence of the peer-to-peer influences.

A detailed survey of the biological research that underlies the stick insect controller can be found in [Crus95], and two other robotic implementations are described in [Müll92] and [Espe93].

II.3.3 Using the Stick Insect Controller on Ariel

This section discusses both the motivation for using the stick insect controller, and the obstacles to its use, on Ariel. Unlike many laboratory robots, Ariel is tested in real world environments. It needed a locomotion controller that would be robust against large perturbations, environmental change, and potentially, even hardware failure. Insects manage to walk despite all these difficulties, so the biological origins of the controller were encouraging.

The crab-like nature of Ariel's locomotion is not an obstacle to using the stick insect controller. The criteria for static stability are the same, regardless of direction of motion.

With only two degrees of freedom per leg, and with all axes of motion being parallel, it may not be obvious how Ariel turns. Ariel turns the same way a tank (with treads) turns. Its turning motion is the legged equivalent of skid steering. Ariel turns by lengthening the stroke at one end of the body, compared to the other.

Ariel's turning kinematics did present a potential obstacle to using the stick insect controller. There was no published experience in modifying the stick insect controller in such a manner. Our experimental experience is described below

II.3.4 The Stick-Insect Controller as a Dynamic System

Simplified systems, and sub-systems

Leg as 1d relaxation oscillator

Theoretically, the stick insect controller is analyzed as a system of coupled oscillators. Each one-dimensional oscillator represents one leg; the full system is six-dimensional. It is also useful to analyze simpler systems, to isolate the behavior of subsystems, or to constrain the entire system to a subset of its possible behaviors.

The one coordinate that describes the state of each oscillator (leg) is periodic. The end of the power stroke is the beginning of the return stroke, and vice versa. One can visualize the leg coordinate as chosen from the points on a circle. Some arc of the circle represents the power stroke and the remainder the return stroke. A multidimensional *phase space* is assembled as the Cartesian product of individual oscillator coordinates.

In these phase spaces, one can label regions as good or bad. In bad regions, two adjacent legs are on their return stroke and the robot is mechanically unstable, as described above.

In the theoretical analysis, the types of questions that we ask concern these issues:

- Desirable Equilibrium — does the system perform repetitive motion on a trajectory that avoids the bad parts of phase space?
- Local Controller Stability — does the system return to a desirable trajectory after being perturbed away from it?
- Global Controller Stability — does the system approach the desirable trajectory from any starting point?
- Information Loss Due to Distribution — since no leg has knowledge of the state of the entire system, is there global information that could be used (e.g. for supervision) that is unavailable to the controller?

We have found results for the following models.

Two-oscillator system, representing two legs adjacent on the same side of the body

There is a good, stable trajectory through the phase space where the rearward leg's return stroke is closely followed by the forward leg's return stroke. This is a desirable relationship between the two legs; it forms the basis of the *metachronal wave*, observed in most real insects. The trajectory is not globally stable — there are starting points where the system passes through bad phase space for several cycles — but the good trajectory does have a wide basin of attraction.

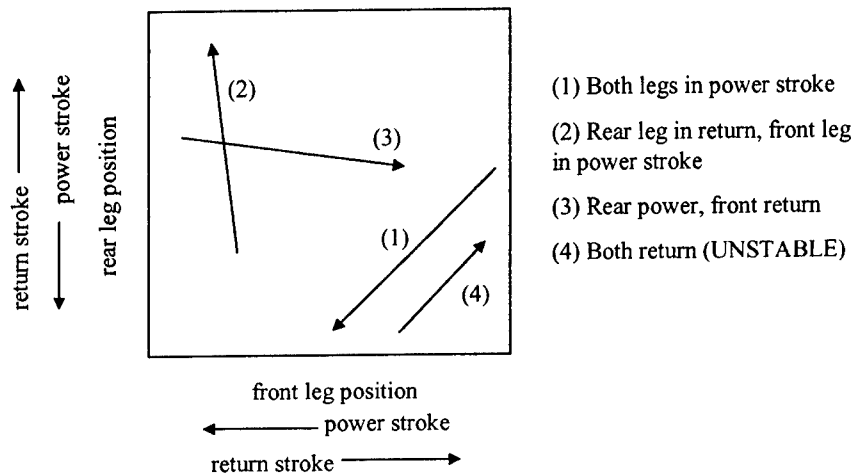


Figure 10: The 2-Leg Phase Space

In Figure 10, The 2-Leg Phase Space, each point in the square corresponds to a pair of positions. The horizontal coordinate represents the front leg position. The vertical coordinate represents the rear leg position. Recall that there is a single constant power stroke velocity for both legs, therefore when both legs are in power stroke the trajectory (Example 1) will be 45° downward and to the left. When the return stroke velocity is greater than the power stroke velocity, if the rear leg is in return stroke and the front leg is in power stroke (Example 2) the trajectory slopes upward to the left at a steep angle.

Similarly, Example 3 depicts that when the front leg is in return stroke and the rear leg in power stroke, the trajectory slopes downward to the right at a shallow angle. Finally, when both legs are returning at the same time (Example 4) the trajectory is upward to the right at 45°.

“Influence 5” [Crus95] extends the rearward leg’s turnaround point as a function of the forward leg’s position

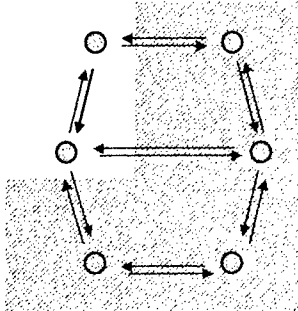


Figure 11: 2 Leg System with 1 Influence

The whole system has 6 legs, 14 influences. This case looks at a subsystem simplifying to 2 legs and 1 influence. The influence results in extending the position of the rear leg before it enters the return stroke. Thus, the phase space is no longer square but rather has an extension representing the delayed return. The fact that this extension happens only for the rear leg (as shown in Figure 12 is a result of the asymmetry of this influence for same-side legs.

The influence modifies the rearward leg’s turnaround point

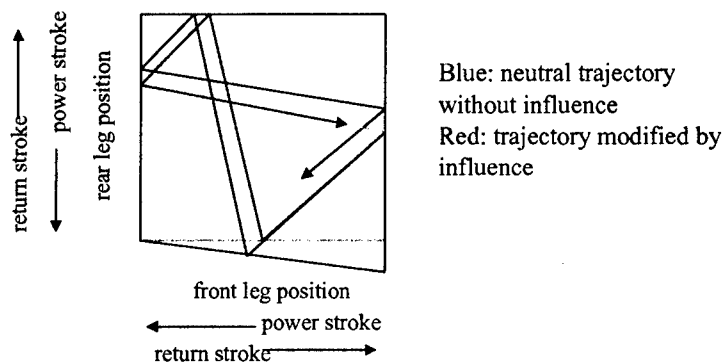


Figure 12: 2-Leg System with Influence 5

The effect of the influence is to make the front leg cycle take slightly longer than the rear leg cycle, therefore the phase relationship between the two legs slowly changes, e.g. of good trajectory moving toward limit cycle.

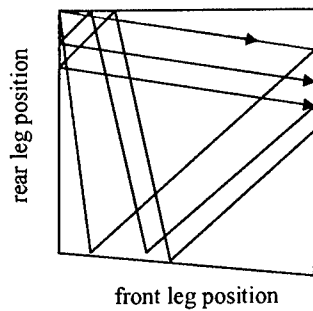


Figure 13: Limiting Behavior for Influence 5

In Figure 13, the stable trajectory under influence 5 has rear-leg return immediately followed by front-leg return. The trajectory converges on a metachronal wave. The stable trajectory can be found rigorously by return-map analysis, but the graphic sketch is roughly correct and quite convincing. In insects, the metachronal wave is a wave of return strokes that moves forward up the body (e.g. it can be easily seen in centipedes). The red trajectory is the limit cycle for a large fraction of the phase space, but not all.

Two-oscillator system, representing two legs directly opposite across the body

There is a good, stable trajectory where the two legs are 180° out of phase. Like the previous system, this trajectory is almost, but not quite, globally stable.

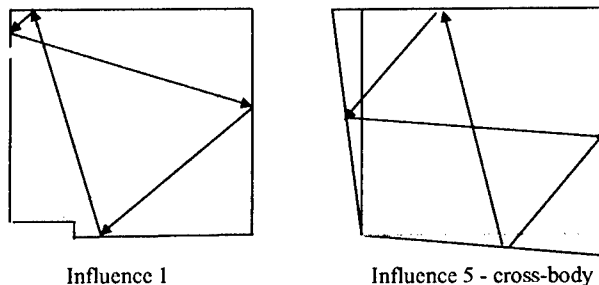


Figure 14: Other results for 2 Leg Systems

In the same-side pair, influence 5 is insufficient to get out of a double-return (bad) trajectory. Influence 1, which changes the rearward -leg's turnaround, will get the system out of a bad trajectory for *some*, but not all, regions of the phase space. In the cross-body pair, symmetric application of influence 5 has a stable trajectory with the two legs 180° out of phase with each other. Note that for the same-side pair, where the legs are not interchangeable, the influences have a front/back asymmetry, unlike the cross-body pair.

Three-oscillator system: each oscillator represents a pair of legs, directly opposite across the body

In this model, we represent all six legs, but assume that cross-body coupling is strong compared to same side coupling. Members of the cross body pairs are assumed to be held 180° out of phase with their

opposite number, as predicted by the above result, and a single oscillator represents each pair. At low velocities there is a good trajectory where the rearmost leg's return stroke leads the middle leg's return stroke, which in turn leads the forward leg's return stroke. This is the *metachronal* wave, as predicted above. There is a bifurcation of the trajectory as the strength of the influence between oscillators grows, eventually leading to mild excursions into the bad regions of phase space.

The same three-oscillator system is also the simplest model where there are *non*-neighboring legs. The return map for this system has been calculated. An important feature is that the map is the sum of the corrections applied by each individual leg. One can prove that the return map of the system where end legs have no knowledge of non-neighboring legs at the other end has a more restricted functional form than that of the system with global knowledge.

Return-map analysis

To reduce the complexity of analyzing the system, we make use of return-map analysis. Instead of dealing with the full, twelve-dimensional system, we simplify to a three-oscillator model. This allows a more intuitive analysis and visualization of what is going on.

Loss of Information from Decentralization

The stick-insect controller, as described above, is composed of locally coupled sub-systems *i.e.* the individual legs communicate with their neighbors. How much information is present in the system, but not available for control because the information is not localized at any one point? The following mathematical model attempts to answer this question.

Another method of representing the phase space of a two-oscillator system (for example, from the above figures) is as a 2-torus (a doughnut). This can be unfolded, as shown in Figure 15 below, to form a square. In this representation, the power and return strokes are distinguishable because of position. In this case, the bad region, that is, where the two legs are both in return, is a smaller, square region. With uniform time spacing, this allows entire trajectories to be represented as straight lines parallel to the diagonal. However, this has the disadvantage that motion beyond the usual turnaround is not represented.

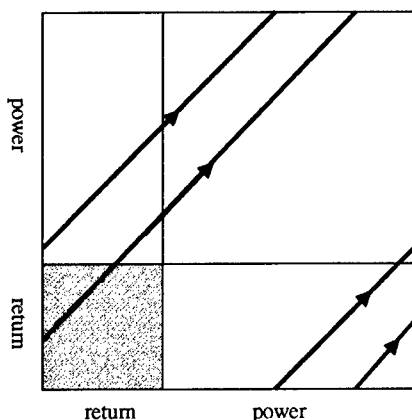


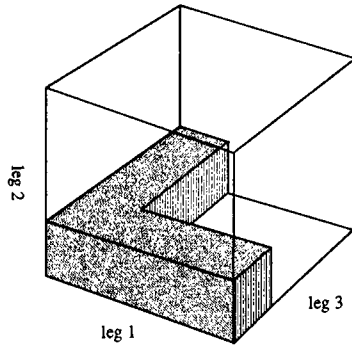
Figure 15: Alternative Phase Space Representation: Unfolded, Uniform Time Spacing for a Two-Oscillator System

Consider a 3 degree-of-freedom system, corresponding to the full 6 legged system, but with cross-body pairs of legs locked 180° out of phase. We separate power and return strokes so that a different range of the coordinate for a leg denotes each one. To keep the geometric reasoning at its simplest, we chose the coordinate so that every coordinate interval represents an equal amount of time.

Note that this configuration space is extremely simplified. Because of the need to connect the end of the power stroke with the beginning of the return stroke, at a single point, we cannot represent the variation of the posterior turnaround point as a part of a continuous trajectory in the space. Instead, we pick a nominal turnaround point and model variation in the turnaround point as a discontinuous jump in the trajectory.

With three cyclic coordinates, the full configuration space is a 3-torus. "Bad" regions of configuration space, where two adjacent legs are in return stroke, form two intersecting square prisms across the space.

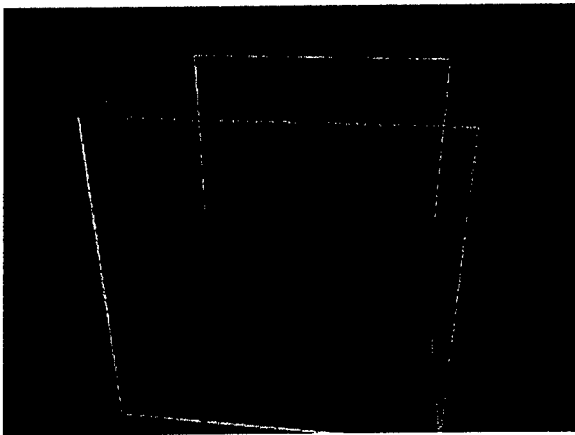
Figure 16 shows the 3-torus unfolded as a cube — in viewing this kind of illustration it pays to remember that opposite faces of the cube are connected. In some cases, when illustrating the periodicity of the space is important, we can place multiple copies of the cube side by side.



Bad regions in RED

Trajectories are
parallel to body
diagonal of cube

Figure 16: 3-leg System shown as a 3-torus unfolded as a cube



An unperturbed good trajectory is very simple in this space, as shown in the figure at right. It is a straight line, parallel to the body diagonal of the cube. Note that this picture supports the standard folklore on legged controllers—the stick insect reactive controller is better at slow motion than at fast motion. At slow speeds, the hole between the bad regions is large, and a wide range of unperturbed trajectories can avoid the bad regions. At faster speeds, the hole narrows until, at the tripod gait, it becomes a single point.

Figure 17: Good trajectory.

Now, let's construct a perturbed trajectory. We model extension of the posterior turnaround by allowing the other (not-turning) coordinates to increase, while the turning coordinate is held at the turnaround point. When the turning leg passes back through the nominal turnaround point, we resume the normal trajectory. The important point about the turnaround is that the non-turning coordinates continue to increase uniformly, and therefore the piece of trajectory is parallel to a face-diagonal.

We will now construct the return-map for this system. Let the three coordinates be x , y , and z , with y denoting the middle leg. Let the starting point of the trajectory be just before an x -turnaround, and let the order of turnarounds be x followed by y followed by z . (We will see shortly that the order does not matter.)

From a starting point $(x_0 = 0, y_0, z_0)$, the coordinates after the x turnaround are:

$$x_1 = 0$$

$$y_1 = y_0 + X(y_0)$$

$$z_1 = z_0 + X(y_0)$$

where X represents the delay introduced at the x turnaround; it is only a function of the y position, because the x -leg is an end-leg with only the y -leg as a neighbor.

Propagating the trajectory across the space from x -boundary to y -boundary, the point before the y -turnaround is:

$$x_2 = 1 - y_1$$

$$y_2 = 0$$

$$z_2 = z_1 + 1 - y_1$$

while the y -turnaround, being the middle leg, depends both on x and z :

$$x_3 = x_2 + Y(x_2, z_2)$$

$$y_3 = y_2 = 0$$

$$z_3 = z_2 + Y(x_2, z_2)$$

Similarly, propagating the trajectory towards the z -boundary:

$$x_4 = x_3 + 1 - z_3$$

$$y_4 = 1 - z_3$$

$$z_4 = 0$$

performing the z -turnaround:

$$x_5 = x_4 + Z(y_4)$$

$$y_5 = y_4 + Z(y_4)$$

$$z_5 = z_4 = 0$$

and finally back towards the x -boundary:

$$x_6 = 0$$

$$y_6 = y_5 + 1 - x_5$$

$$z_6 = 1 - x_5$$

yields the return-map:

$$y_6 - y_0 = X(y_0) - Y(x_2, y_2)$$

$$z_6 - z_0 = X(y_0) - Z(y_4)$$

Substituting for y_4 :

$$y_6 - y_0 = X(y_0) - Y(x_2, y_2)$$

$$z_6 - z_0 = X(y_0) - Z(y_0 - z_0 + Y(x_2, y_2))$$

and for x_2 and y_2 , using

$$x_2 = 1 - y_0 + X(y_0)$$

$$y_2 = 1 - y_0 + z_0$$

Software Implementation

In this section, we discuss the practical implementation of the stick-insect controller.

Linear Controller with Trajectory

The lowest level of software implements a linear feedback controller for each individual joint. Ariel's joints (6 hips and 6 knees) are each controlled by a standard linear position controller. The position controllers run at 60Hz. The basic dataflow for one controller is shown below.

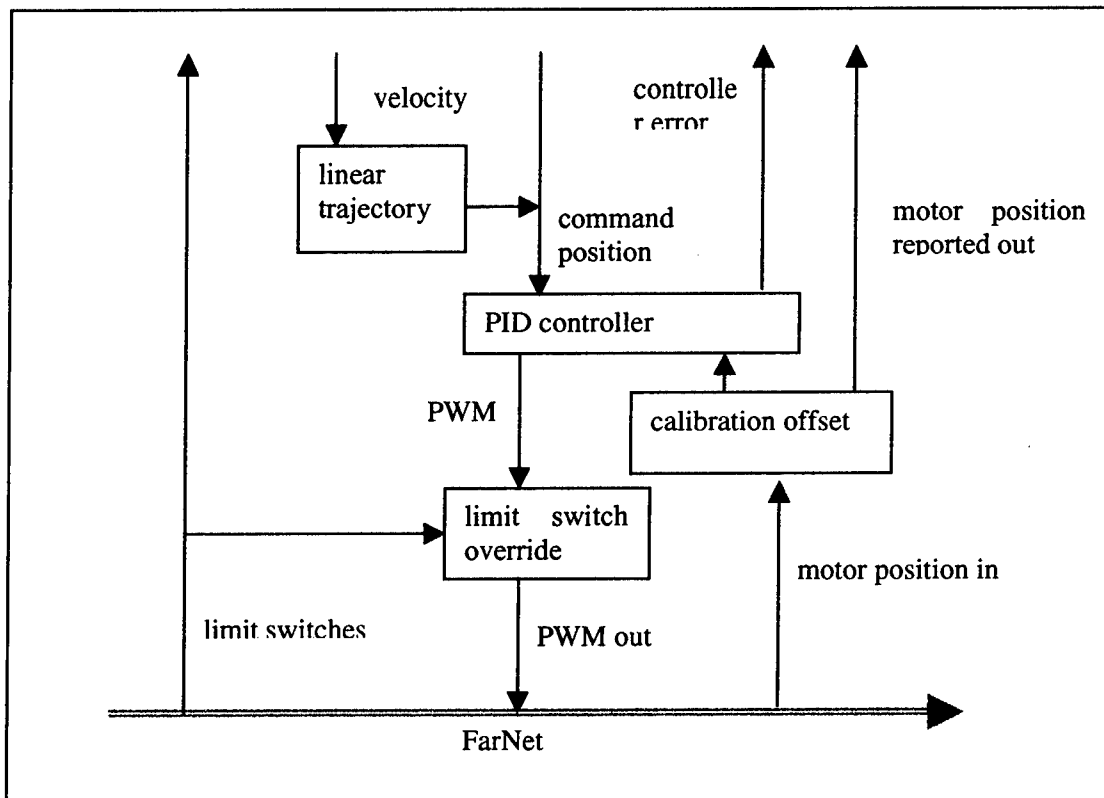


Figure 18. The linear controller (one per joint — 12 total)

Each controller has:

- PID controller with integral decay;
- sets of PID coefficients selectable from software *e.g.*, compliance setting in different situations;
- simple trajectory (*i.e.*, 1-D velocity);
- reporting of controller error for higher-level diagnostics.

Note that the PID feedback loop is closed over FarNet.

Leg Cycle

The individual leg cycle is composed of coordinated motions by hip and knee joints. A single bit of state distinguishes power stroke from return stroke.

The power stroke consists of a constant velocity trajectory by the knee joint, and a simple height-correction by the hip joint. We would like the foot to remain at a constant vertical distance from Ariel's body as it sweeps horizontally. If θ denotes the hip angle from horizontal, and ϕ the knee angle from vertical, u the length of the upper leg, and l the length of the lower leg, then constant foot height is given by:

$$u \sin \theta = l(1 - \cos \phi).$$

If the hip is typically positioned near 0° while the knee swing is centered around 90° , then we can approximate to second order in the angles. Canceling u and l , because Ariel's upper and lower legs have approximately the same length, we employ the following correction to hip height:

$$\theta \approx \phi^2$$

which is, obviously, cheap and efficient to implement.

The return stroke needs no correction in height. We do anticipate, however, the end of the return stroke (the anterior turnaround point) and start the hip descending *before* the turnaround point is reached to arrive at power stroke height at the end of the stroke.

hip angle θ

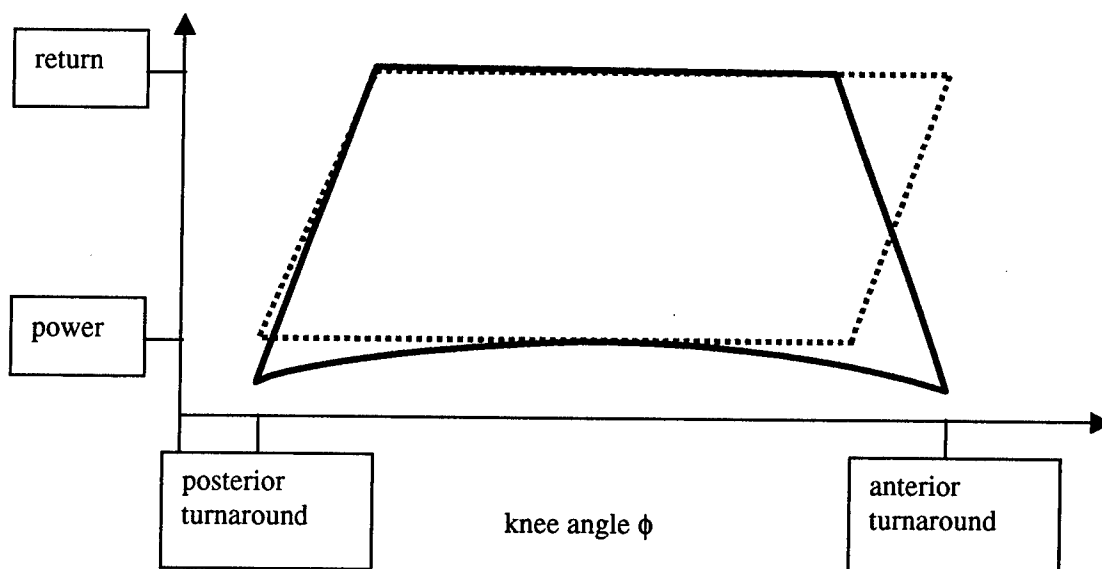


Figure 19: The trajectory.

Influences

Each leg, executing a gait cycle as described above, makes available two pieces of information: a bit describing whether it is in power or return stroke, and the fraction (between 0.0 and 1.0) of the stroke. This information is transmitted to neighboring legs, where it forms the input to influence calculations. (Note: this is unlike the biological model, where influence is "calculated" in the transmitting leg.)

Control from Upper Levels—Implementation on the mini68332

How does the stick insect controller fit in a larger robot control architecture? “Open loop” commands and tuning parameters are simple. Recognition of global stability and fault intervention is harder.

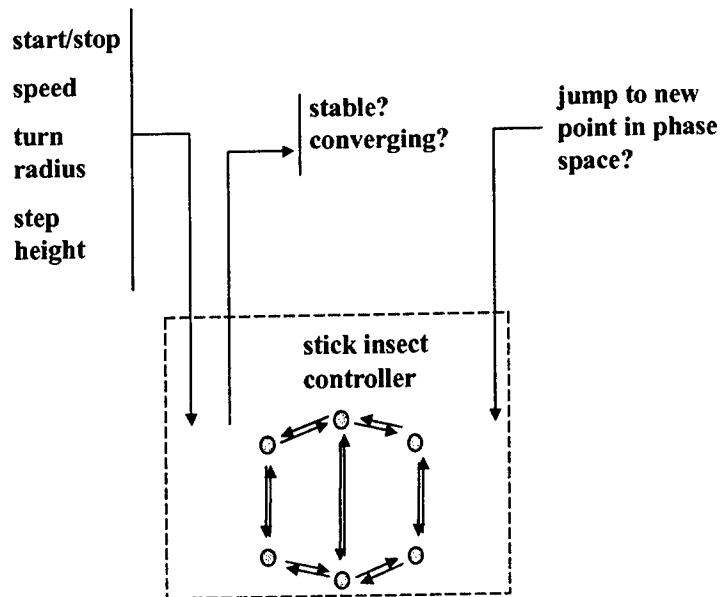


Figure 20: Upper Level Control

The stick insect controller is really six loosely coupled controllers (one for each leg). The ease of downward control and the difficulty of the other things follow because of this. Control parameters that map directly onto parameters of individual leg motion (e.g. velocity) are easy to deal with. On the other hand, control parameters that are combinations of more than one leg's properties are not localized anywhere (e.g. do we have a stable tripod of legs on the ground?).

Inversion Layer

Communicating directly across the mirror to the controllers on the rFlex processor, the purposes of the inversion layer are to provide joint and leg abstractions (“objects”) on the miniboard side of the mirror; and to present an Ariel that is always “right side up” to higher levels of software.

Since Ariel can walk and operate upside down, all operations such as walking, or even just standing, need to work in both orientations. The inversion layer hides the detail of which way up Ariel currently is, and allows higher-level software to be written in terms of the normal orientation only. By calibrating the joint coordinates so that zero is horizontal (“straight out”), the transformation of coordinates is particularly simple. If this layer detects a change in orientation, it re-issues most recent per-joint or per-leg command as a low-level recovery procedure. This doesn't prevent the implementation of higher-level behaviors in response to inversion, but provides a simple robust lowest level. Inversion affects the following:

- joint position and velocity — inversion negates positions and velocities
- gait parameters — inversion negates turnaround points, and power and return heights.
- compasses — inversion rotates the magnetic field and gravity vectors 180° around the local x-axis, and interchanges the two compass' readings;
- walking controller — inversion reverses which direction is considered a positive rotational velocity.

Sensing of Ariel's inversion status is done by a mercury switch mounted on the I/O board, and by the two compass/inclinometers. A small behavior provides integration of these sensor inputs and low-pass filtering for the mercury switch input (the mercury switch is more prone to triggering due to vibration while walking).

Walking and Posture Supervision

The purpose of the walking supervisor on the mini68332 is to invoke and monitor the stick insect controller on the Coldfire. The walking supervisor calculates stroke lengths and velocities for individual legs. (When Ariel turns, not all legs have same gait parameters.)

Posture supervision informs the walker, and other behaviors, of appropriate leg-heights in order to achieve some goal. With inclinometer input, the posture supervisor can choose a goal of keeping the body approximately level. With input about local flow conditions, the posture supervisor could "lean into" the flow to resist it.

Mission Component Behaviors

At the next level, just below the top level of control, are the behaviors that comprise the components of a mission. They are:

search: a guided walk, servoing on direction;

calibrate: before the robot can do anything else, it must calibrate joint positions;

station keeping: triggered by detection of a mine, stop and raise a leg to indicate success.

Mission Arbiter

Ariel's missions are comprised of several sub-behaviors that are run either concurrently or sequentially, depending on the action performed by the behavior. For instance, the behavior that causes the robot to stand up is run once at the beginning of the mission and then only run again if the robot finds itself not standing again later. The posture supervisor, however, is intended to run continually regardless of what behavior is in control, advising all relevant behaviors about leg heights necessary to maintain a particular posture. The latter type of behavior simply needs to be started at the beginning of a mission, but the former behaviors need to be started and stopped carefully to avoid having more than one running at once and competing for actuators. This mission monitor behavior is responsible for this.

The purpose of the mission arbiter is to let the user write a sequential plan of robot actions. The mission monitor behavior is a state machine that takes a list of behavior names and runs through them in order until it reaches the end, at which point the mission is declared successful. It knows the details of how to activate each behavior, what signals to look for that indicate task completion (and whether the task was successfully completed), and how long to wait for that signal before declaring that something has gone wrong and taking some other action. In the case of the mission displayed at the demo, if a behavior takes too long to complete, the monitor simply sets a failure flag and stops the mission. It is also responsible for selecting between behaviors given the completion status of another behavior. For instance, if the search behavior is completed successfully, the mission monitor starts the station keeping behavior, whereas a failed search simply signals mission failure and stops the robot.

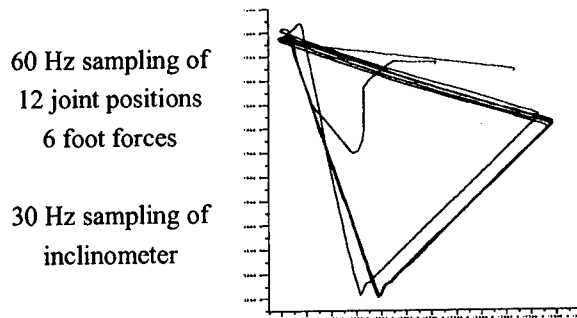
II.4 Results

II.4.1 Straight Line Motion

As described above, the stick insect controller was implemented on top of basic linear position controllers for Ariel's 12 joints. The individual gait cycle was shaped to give the foot straight-line motion during the power stroke, for a smoother walk, but a natural arc, for simplicity, during the return stroke.

The strength of the inter-leg influences was tuned for maximum resistance to perturbations. While other reports [Espe93] have claimed a wide range of acceptable parameters for emergence of a gait, we found that the parameter range that yields a gait with the best resistance to perturbation is significantly narrower.

Ariel was tested walking on land, and underwater in surf conditions. The controller worked well at varying speeds up to a tripod gait at approximately 30cm per second.



At full speed, there is some lateral lurch just before the two leading corner feet touch down. We speculate that this may be related to bifurcation in the 3-oscillator system, but this has not been confirmed.

Figure 21: Real world data: Leg Position for Straight Motion

II.4.2 Turning Motion

The basic controller was extended for turning by varying power stroke speed along the robot, and scaling power stroke length proportionally so that time duration for a power stroke remained constant from leg to leg.

Again, this was tested walking on land, and underwater in mild surf conditions. Ariel is able to perform direction servoing using compass input and walk a search pattern composed of straight line passes. The turning gait works well when the turning radius is large compared to the inter-leg distance, *i.e.*, with the center of turning well outside the robot.

Turning with the stick insect controller does not work well when Ariel is called upon to turn in place, *i.e.*, with the center of turning more or less under the robot. This is a consequence of an incompatibility between the skidding nature of Ariel's turning kinematics, as described above, and the cross-body 180° phasing that is the behavior of the stick insect controller.

We have implemented an alternate turning controller. This alternate controller executes a three-phase cycle, where all four corner legs push simultaneously in one phase, and then two additional phases are used to reposition the corner legs for another power stroke. During the main power stroke, the middle two legs retract and play no part; they descend during the other two phases to stabilize the robot while the corner legs return.

II.5 Conclusions

The stick insect controller is a functional piece of control software that originated in biological research. It performs well on a robot deployed in a real world application. The simplicity of the original stick insect controller was advantageous for the software engineering process. We have extended the stick insect controller to the crab-like motion and skid-turning kinematics of the Ariel robot. The extension performs well for large radius turns, but performance degrades as the radius of the turn gets smaller. A substitute turning controller makes up for the limitations in small radius turning. The distributed nature of the stick insect controller appears to incur some information loss, compared to a controller with centralized, global, knowledge. Return-map analysis provides some insight into this information loss, and can be the basis for supervisory control that uses this missing information. Alternatively, the return-map analysis may form the intuitive basis for a more probabilistic method for supervisory control.

II.6 References

- [Crus95] Cruse et al. "Walking: A Complex Behavior Controlled by Simple Networks." *Adaptive Behavior* Vol. 3, No 4, pp 385-418. 1995.
- [Barn98] D. Barnes. "Hexapodal robot locomotion over uneven terrain." Proceedings of the IEEE Conference on Control Applications. Trieste, Italy. September 1998, pp 441-445.
- [Espe93] Espenschied et al. 1993. "Leg Coordination Mechanisms in the Stick Insect Applied to Hexapod Robot Locomotion." *Adaptive Behavior* Vol. 1, No 4, pp 455-468.
- [Isid95] A. Isidori. "Nonlinear Control Systems" Springer Verlag. 3rd edition, 1995.
- [Müll92] Müller-Wilm et al. 1992. "Kinematic Model of a Stick Insect as an Example of a Six-Legged Walking System." *Adaptive Behavior* Vol. 1, No 2, pp 155-168.
- [Yip89] Kenneth Man-kam Yip. KAM: Automatic Planning and Interpretation of Numerical Experiments Using Geometrical Methods. MIT AI Technical Report 1163, August 1989.

Section III. CLIN 0004: Adding an Active Vision Head to the M4 Robot

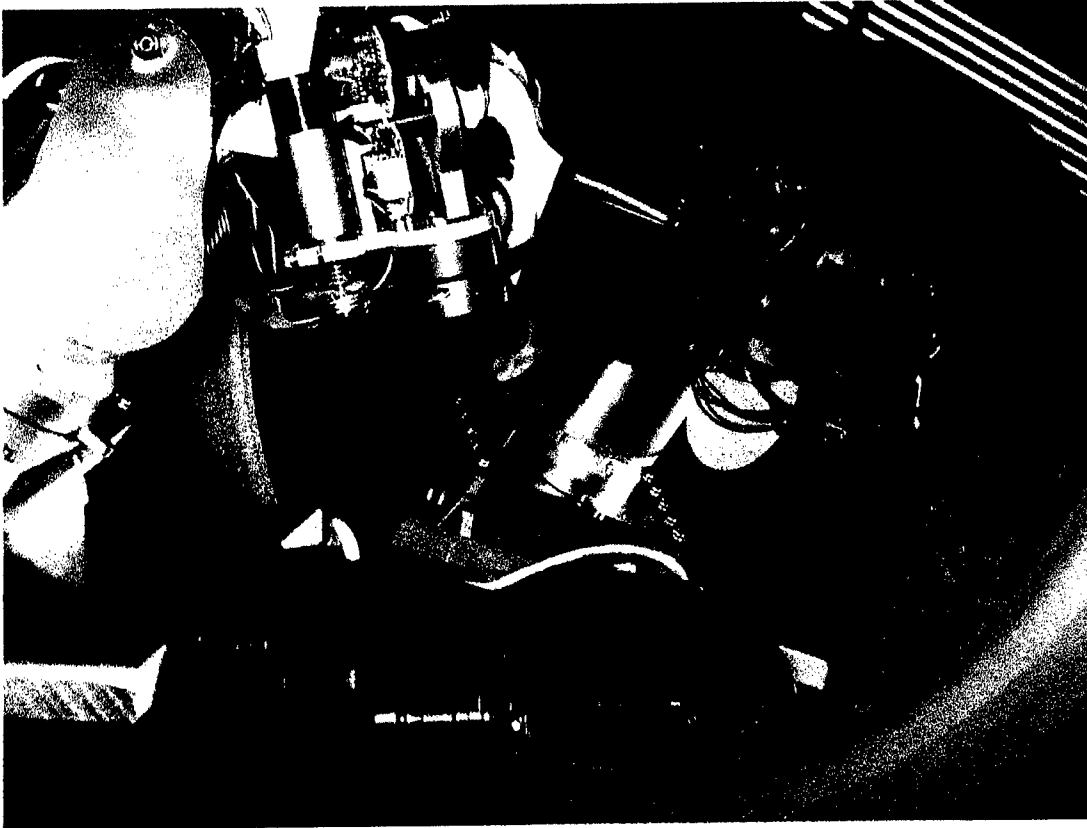


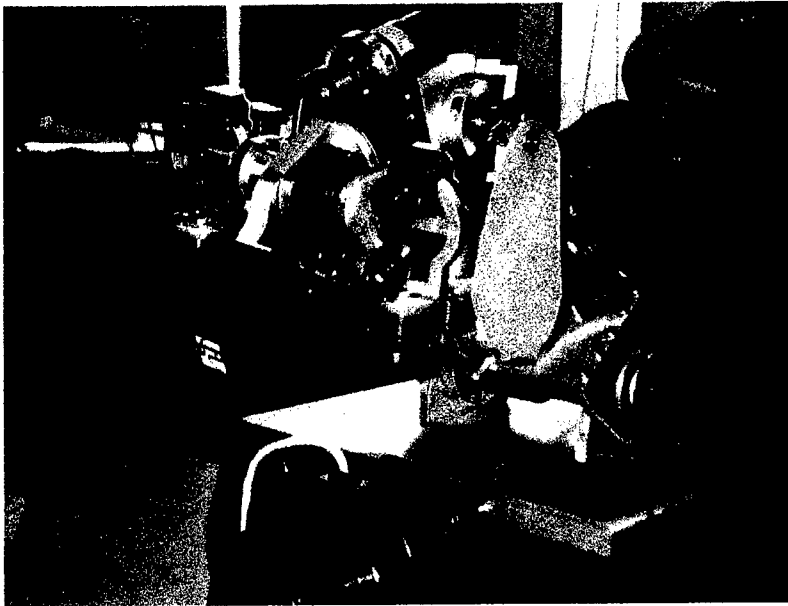
Figure 22. Macaco - M4/M2 Project. Rodney Brooks and Artur Arsenio. Primary Sponsor: Air Force Aerospace Research – OSR. Secondary sponsor: iRobot (IS Robotics) contract number F30602-96-C-0280.

III.1 Introduction

This final report for CLIN 0004 is an extension and modification of the report previously submitted on November 30, 2000. In particular, we have added sections for:

- Integration of the Macaco M4/M2 head with the body
- The new primate appearance for the Macaco head
- Stereo Vision mechanism to support navigation

This document reports research for the Macaco-M4 project – head and brain development for a mobile robot in joint collaboration with the MIT Leg Lab. Macaco is a 7DOF small and light robotic head, with four color CMOS cameras and one thermal camera. This head will be incorporated into a moving body. The merging of social competencies with navigation capabilities requires an architecture that integrates all the modules coherently, such that Macaco is given a personality demarcated by curiosity and a wish of interacting with people, together with a strong instinct for safety.

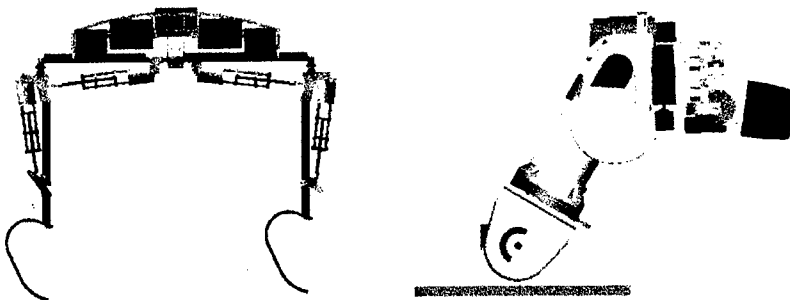


One approach of AI is the development of robots whose embodiment and situatedness in the world evoke behaviors that obviate constant human supervision. With this in mind, Macaco was designed for navigation in unstructured environments. Thus, its vision system must comprise methods of assessing the constantly changing terrain. Navigation for obstacle and potentially treacherous landscape avoidance, slope detection, and gaze stabilization are all requisites for such competence. Furthermore, in order to be a convincing social participant, the vision system must also allow for person detection and inference of human gaze direction. All of these vision modules must also be accessible to each other and concurrent with other sensor input from the rest of the body.

III.2 Body/Head Integration

The design and construction processes of the M2/M4 Macaco head took into account severe restrictions on weight, size both at the mechanical and hardware levels. These restrictions were taken into account so that the body could support the integration of the head in latter stages.

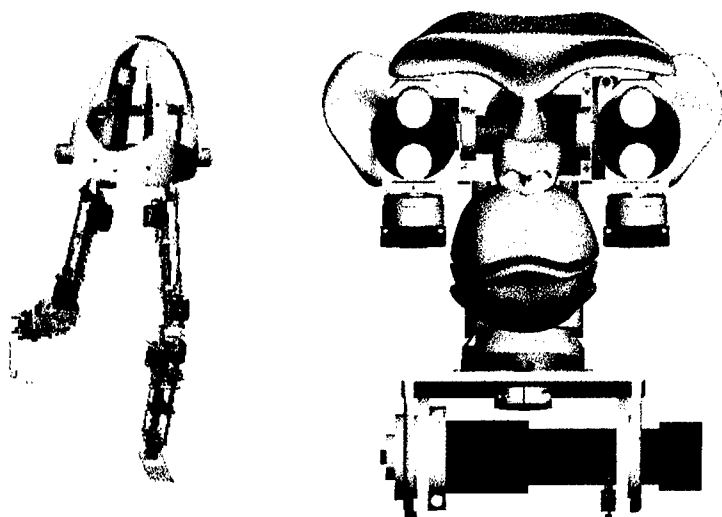
In addition, the aesthetics of the head and the one of the body matches. Thus, the original design of a dog's like body inspired also a dog's like head, so that both systems would integrate perfectly as an all.



Since one of the goals was the design of a head that could match both quadruped or bipedal bodies, the design of the head was made flexible, and a small number of changes allowed a primate like look for the head, in order to match the body of M2. The mechanics of the body allow the integration of the head, due

to its light weight, small size and easy connection at the base of the neck. Indeed, since both the body and head are equipped with gyros (for different reasons: the gyros are used at the body for posture stabilization, while at the neck they are used to support visual stabilization and ego-motion), the integration of both systems does not require special hardware integration, besides a small bandwidth communication channel from the head to the body to send information concerning terrain slopes, direction to follow, and obstacles requiring a stop or stairs.

Thus, a serial link for communications and eight screws for fastening are all that it takes to connect both systems.



III.3 Hardware Architecture

After a detailed study of the computational requirements for such a system, the hardware was selected and organized as shown in Figure 23.

- Three half-size (7.35in x 5.0in) HS6200 computer boards
 - Initially was intended a PIII800MHz –speed had to be reduced to 600MHz
- Two PC104plus (3.6in x 3.6in - stackable) computer boards
 - K6-2 400MHz
- All boards with Internet 10/100Mbits and VGA

Communication among the modules is implemented by a network of six Ethernet cards. Additional Ethernet cards may be used for increased bandwidth.

This flexible network incorporates one server, which is connected to a notebook hard drive (removable), while one of the clients, which is connected to an 112Mbytes DiskOnChip can also operate as server if required.

The operating system running on all the processors is OS-QNX, selected because of high performance, the possibility of portability for the existent software in our lab, and real time transparent communications.

A dog like design

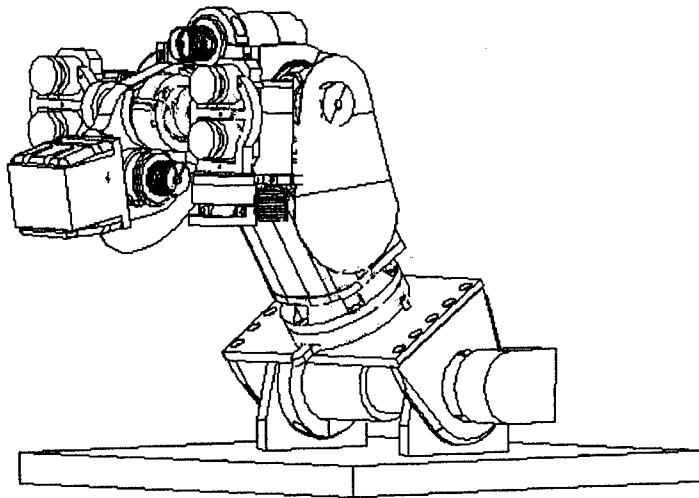


Figure 24: M4 (dog-like head) design.

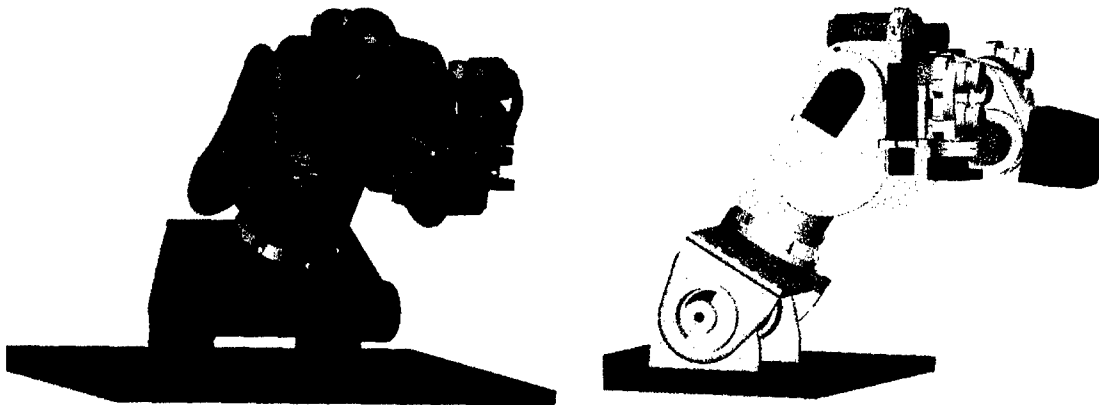


Figure 25: M4 solid model.

A primate appearance

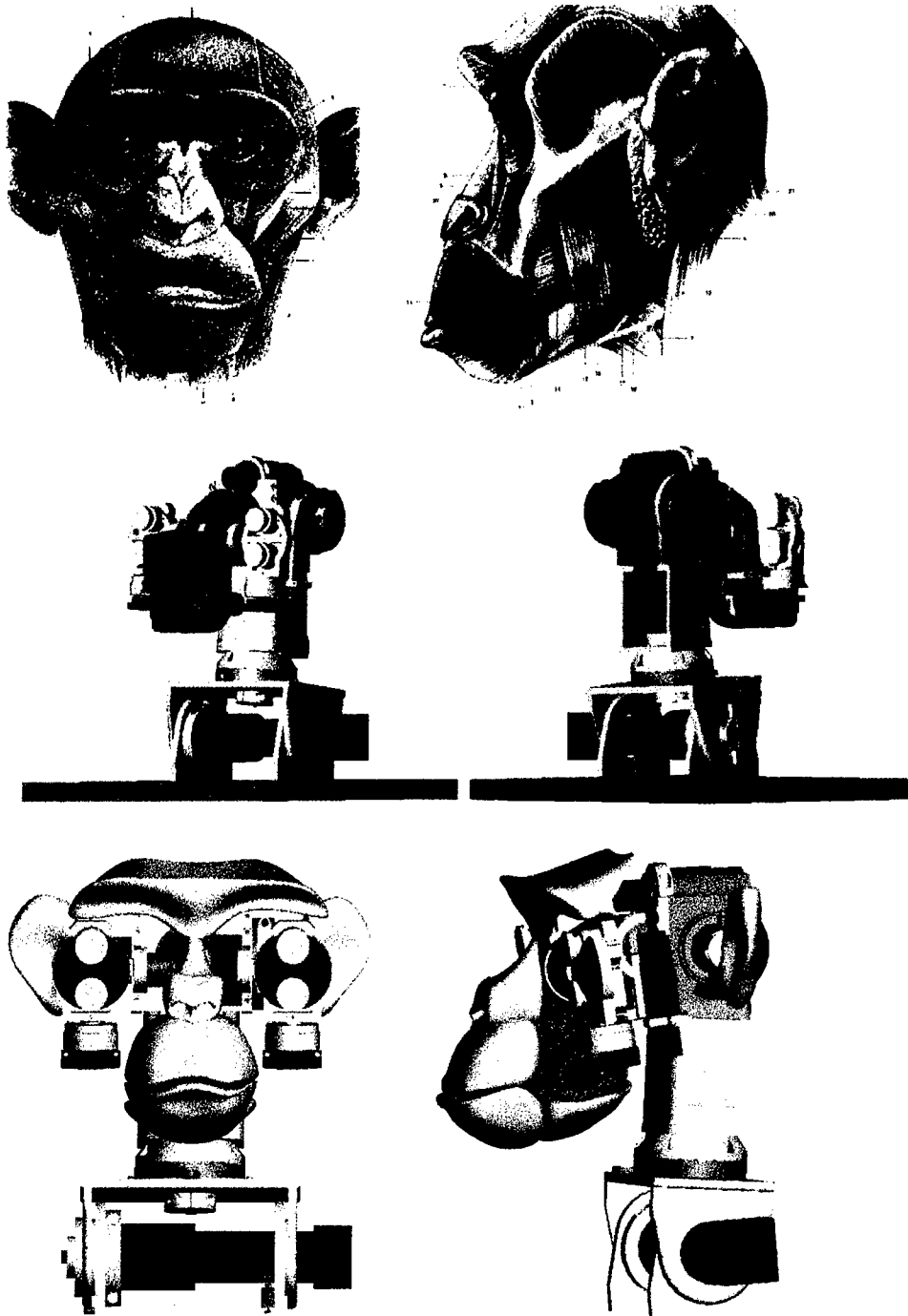


Figure 26: M2 (primate like head) design, with four parts replaced, and a stereolithography artwork. Top: Chimpanzee face anatomy. Middle: Primate head skeleton (four parts replaced). Down: Final primate design.

The Macaco/M4 head was manufactured in *Aluminum* and a small number of parts in Delrin. The head is small, light, and includes inertial sensors, CMOS and thermal cameras, seven motors and equal number of encoders. All the parts are referred on the following figure, and described next.

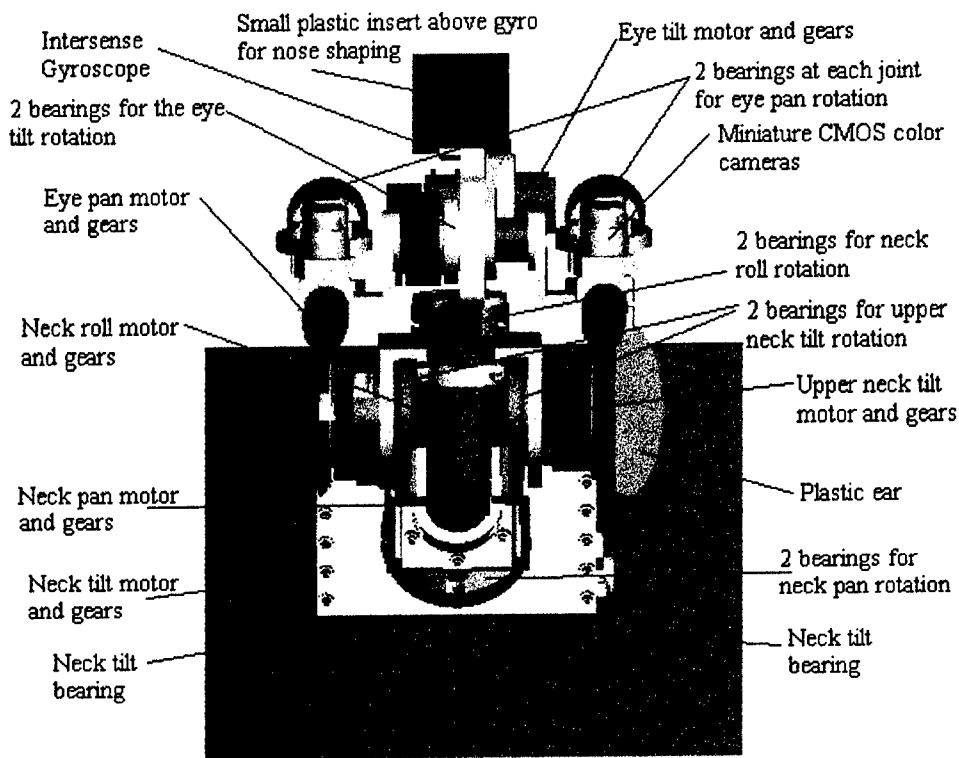
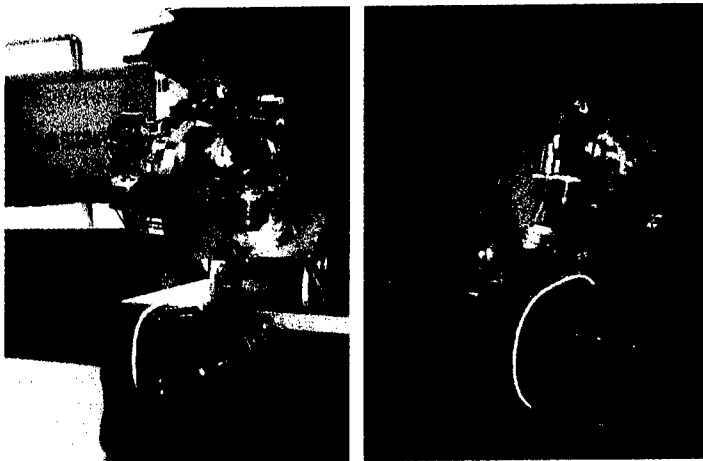
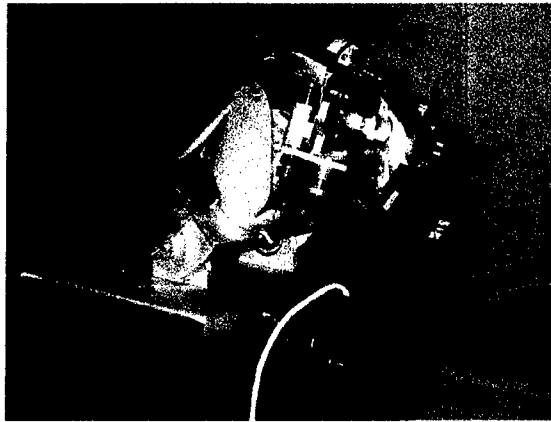
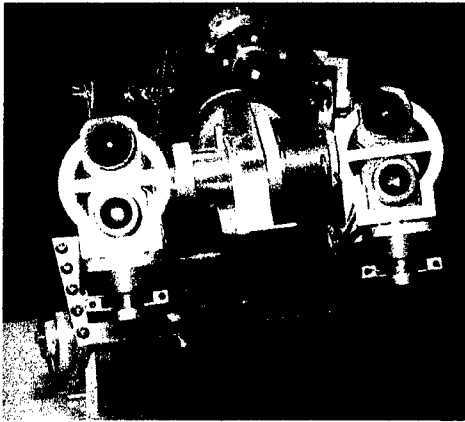


Figure 27: M4/M2 Components.



- Head, motors & sensors:
- Weights ~3.7 lbs

A trade-off between motors power and head weight is difficult to solve because actuator power available is not continuous. The actuators selected consists of *Maxon motors (6W and 20W)* and *MicroMo motors (3.8W and 1.4W.)*



- Hardware:
~1.4 lbs each PIII board, ~0.7 lbs each K6-2
~0.3 lbs each framegrabber, ~0.2 lbs each motor controller/amplifier
~0.22 lbs 20GB notebook hard drive
1 small rack hand-made

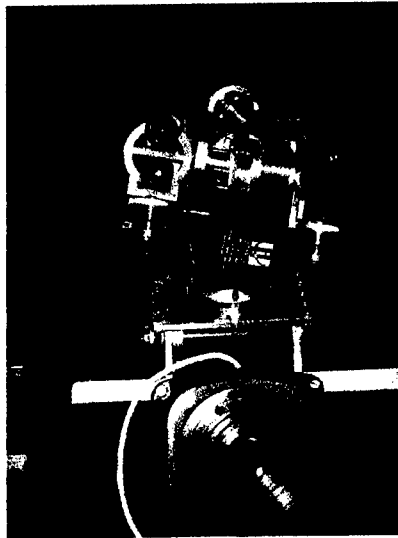
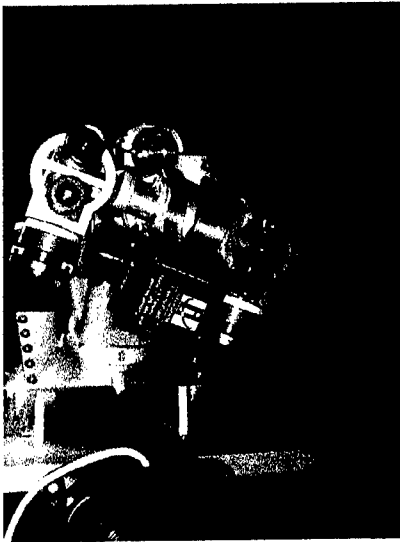


- Power supply (computers and motors):
Together with video amplifiers, video to PC converter and Internet hub requires another rack, which is removed upon integration with the body.

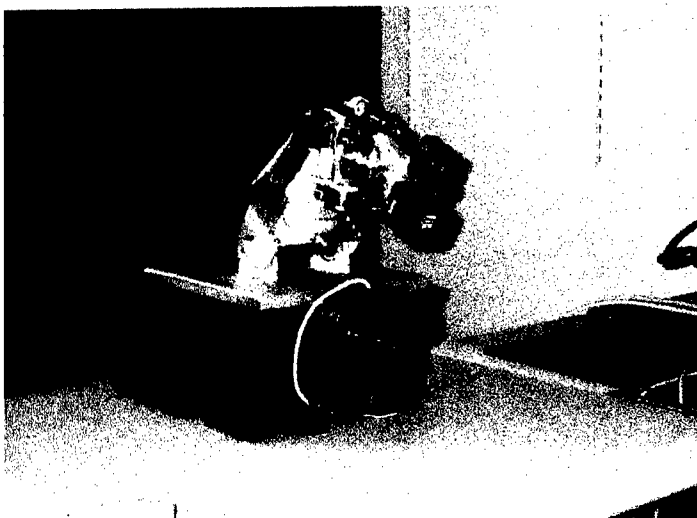


Four moving CMOS miniature color cameras

- 2 foveal: high resolution: $31^\circ \times 23^\circ$ view, 8.0 mm lens
- 2 wide: low resolution: $110^\circ \times 83^\circ$ view, 2.1 mm lens



- No fixed cameras - egomotion will always be present anyway - moving creature!
- CMOS technology better for dynamic scenes
- Thermal camera for night navigation & people detection



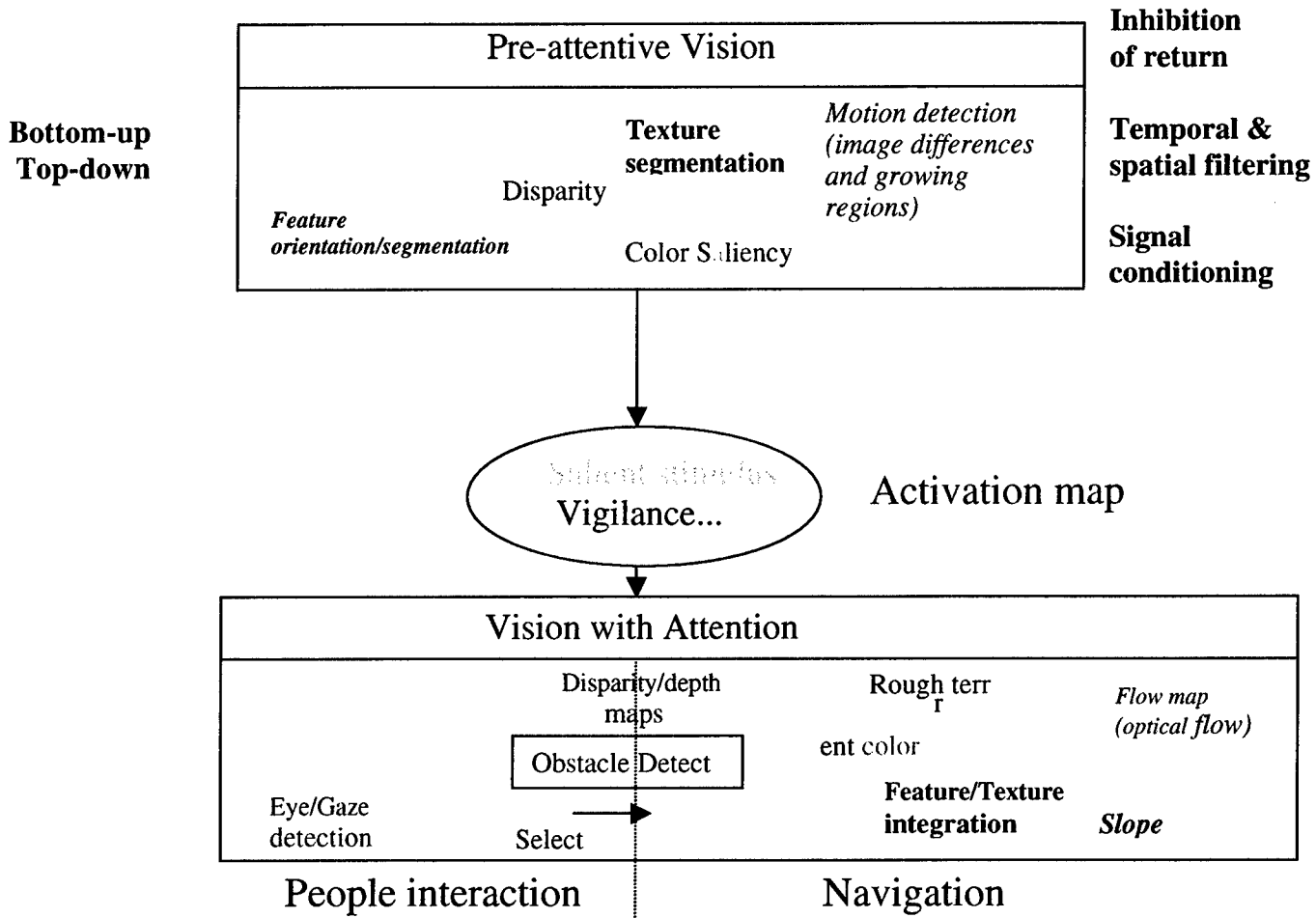
- Distance between the eyes ~3.5in
- 7 degrees of freedom
- 4 in the neck
- 3 for the eyes
- Inertial sensor for VOR (only 3D in rotation).

III.4 Software Architecture

Focus on navigation and social interaction

The software architecture being developed consists mainly of three macro structures: one reflexive, automatic; another that requires dropping attention to certain events; and another that can select the events to which attention will be dedicated and which takes the higher level decisions, such as choosing directions, based on the internal state of the robot, the stimulus received and an embodied simulation of possible events of actions.

- Visual attention mechanism



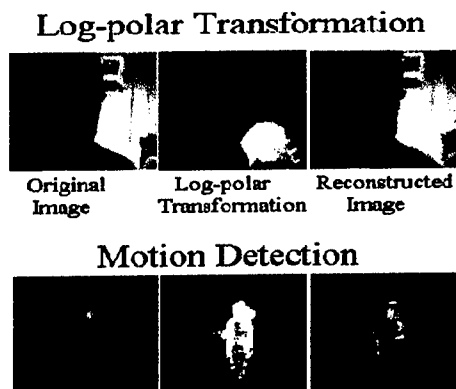
How will the robot share the attention between multiple targets? Although several people may be detected simultaneously, the Face Detection routine will require a decision by the system as to which person should be selected. Furthermore, the vergence process will require decisions on each direction to look, and the stabilization and control decisions on each kind of motion should be applied each time. Attention to a certain kind of stimulus will be dictated also according to the state of the robot and the task being performed. In fact, the robot will be looking constantly for salient stimulus that may be navigation related, such as presence of obstacles, or for more general stimulus, such as movement, people or face detection. Attention mechanisms are already implemented at our lab, [40]. An attention mechanism is being developed to satisfy the functionality and necessities of a mobile platform.

- Motivation drives
 - Curiosity
 - Safety
 - Social interaction
- Releasers from body sensors
 - legs
 - inertial sensors
- Competing behaviors
- Plasticity

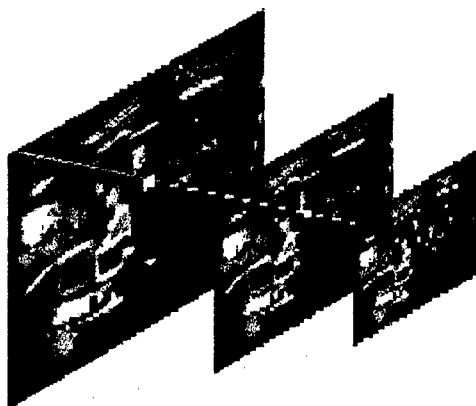
We are also trying to simulate the plasticity in the nervous system, which will increase the flexibility and capabilities for learning and performing the desired tasks.

Modules implemented currently under M4 project:

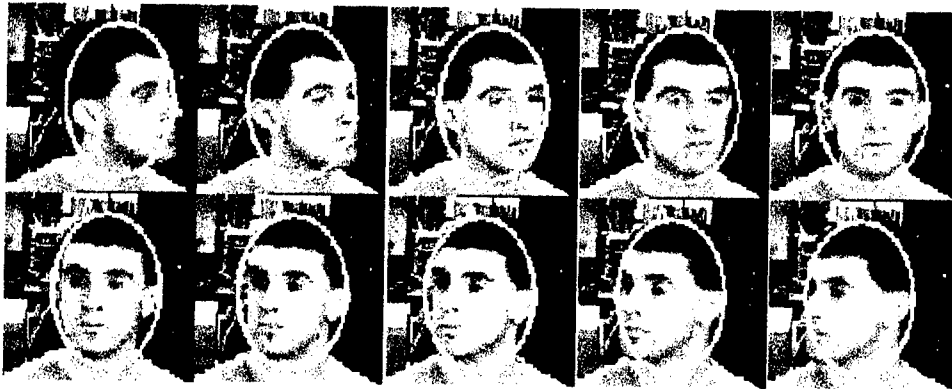
- Feature orientation
- Motion detector
- Log polar transformation of the image input



- Disparity maps
(with Multi-scale resolution)



- *Face detection:* Oval models - looks for a partial ellipse based on edge gradients to detect a variety of head orientations



- Control Oculo-Motor

Two versions, implementing VOR, saccades, smooth pursuit, vergence

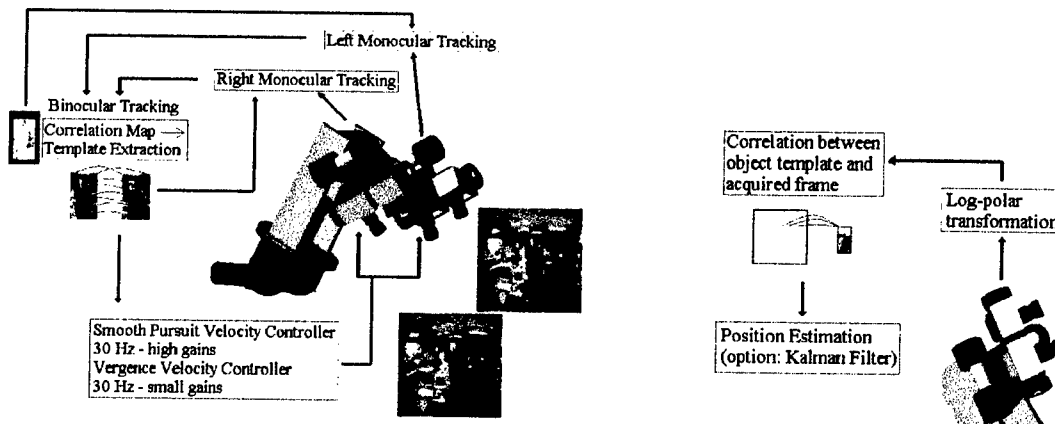


Figure 28: Active binocular tracking (1st version).



Figure 29: Smooth pursuit tested on Cog.

- Optical flow maps
- Sensory/Motor coordination and Stabilization

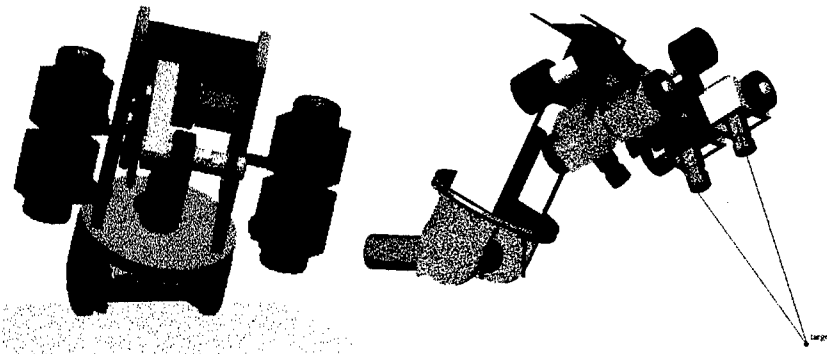


Figure 30: Approaching postures (curiosity behaviors - left), and object tracking (right).

- Stereo Vision for Navigation

Depth information is extracted from a scene using a binocular geometry. By matching correspondent points in two calibrated cameras, one can obtain 3D information from the environment.

For calibration purposes, it was first used a linear calibration algorithm. After several experiments, it was decided the use instead of a nonlinear calibration procedure, since the software for the latter is available on-line from Intel OpenCV.

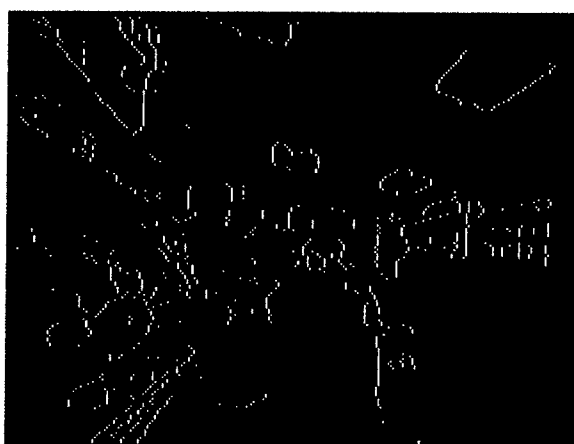
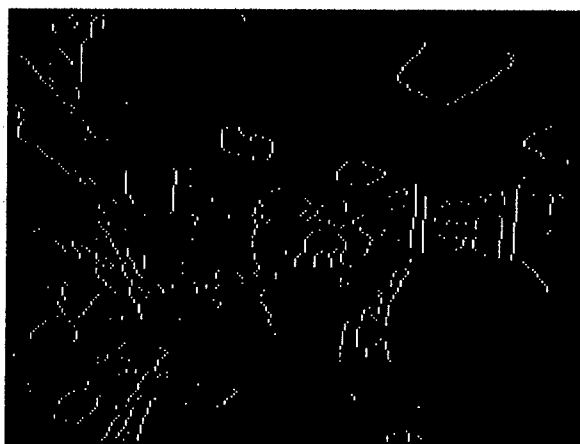


The first step of the stereo algorithm is to compute the epipolar lines, and then to rectify both images in such a way that the epipolar lines become parallel, and thus the correspondence process is performed by matching a line in the left image with the same line on the right image.



Feature-based methods such as dynamic programming require the extraction of contours. Several techniques were compared: Sobel masks, gradient masks, and the Laplacian of a gaussian. The performance of these edge detectors depend in great extent on the scenes and goals:

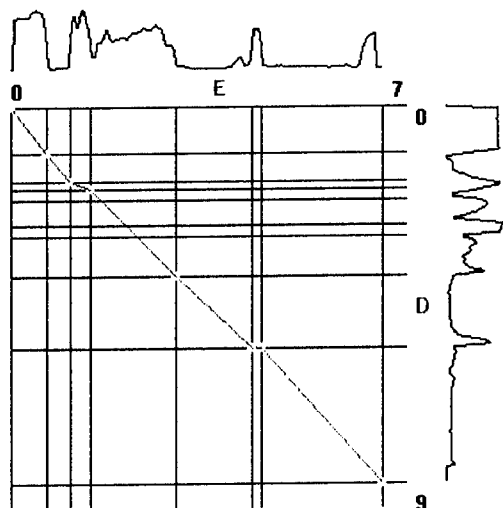
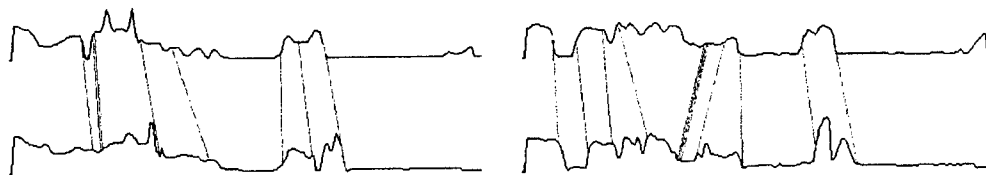
- Sobel masks produce double edges (although procedures are available to reduce the results to single edges, but which increase computational efforts). Good when smoothing of the image is desirable. This algorithm is fast, and gives both gradient and orientation information.
- Gradient masks: very fast, but gives poor results. Gradient masks are very sensitive to noise.



- Laplacian of the gaussian (also known as Marr-Hildreth edge detector): Gives the best performance, but is computationally expensive, because requires the convolution of a 2D gaussian with the entire image. The process was speed up by using an approximate method to compute the convolution that reduces by a factor of 10 the computational effort.

Several matching methods were experimented:

- Correlation: produces dense maps, but it is computationally heavy.
- dynamic programming: the fastest algorithm, although matches were available only for a small percentage of the image (the edge contours). It provides the best results.



- relaxation method: consists on using constraints to guide the matching along several iterations. A good compromise between correlation and dynamic production, both in terms of speed and dense maps.
- techniques combining several methods: It consists of using one method at a coarser level and then to refine using the same method or other method. No significant improvements.

A linear algorithm for minimizing the square error was applied for the determination of 3D information from a matching.

Stereo Vision and Navigation

Information concerning the location of world objects relative to the robot is important for navigation. Indeed, from this information can be extracted the following:

- Slope of the terrain – combined with the gyros information, provides the body with a measure of the slope of the terrain. Depending on this slope, another direction may be selected, or else a different posture of the body may compensate the slope.
- Obstacles: Direction of movement is selected so that robot is guided to free areas.
- Soft terrain: Patterns on the 3D data can be used to extract information concerning features of the terrain.
- Climbing stairs: additional processing will allow the identification of clusters of special patterns, such like stairs

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

*The advancement and application of Information Systems Science
and Technology to meet Air Force unique requirements for
Information Dominance and its transition to aerospace systems to
meet Air Force needs.*